كتاب في البرمجة بواسطة لغة السي بلس بلس



كـتــــــاب

الإكسيــــر

نسـخـة إلكترونيــة مجانيـــــة

البرمجة الكائنية ، القوائم المترابطة ، الملفات ، الاستثناءات ، القوالب . . . . .

OOP ,Linked List , Files, Exceptions , Template …..

Elixir In C++ Language

سلطان محمـد الثبيتـي

1426هـ

الفهـــرس

الموضوع	رقم الصفحة
المقدمـة	8
1–الوحدة الأولى أساسيات السي بلس بلس	12
الخطوة الأولى	12
الخطوة الثانية	14
الأساسيات	16
المتحولات أو المتغيرات..	16
أنماط البيانات وحجومها..	16
الثوابت	17
الإعلانات والتعاريف	17
العمليات الحسابية	18
عمليات المقارنـة أو العلائقية..	18
التعابير وعـملية الإسناد..	18
التعابير الشرطية	18
عمليات الإنقاص والزيادة..	19
المعامل sizeof	19
القراءة (الإدخال) والكتابة..	20
مساحات الأسماء	20
التعليقات	21
مثال (1)	21
مثال (2)	22
الثوابت الرقمية	23
التوابع	23
2– الوحدة الثانية: بنى التحكم	25
بداية	25
الجملة if	25
الجملة if/else	26
الجملة else/if	27
مثال عـملي	29
الجملة switch	31
استخدام المعاملات المنطقية مع الجملة if	34
المعاملات المنطقية	35
مثال عـملي	35
الجملة goto	38
الجمع التراكمي	39
الجملة do/while	40
مثال عـملي	40
الحلقة while	41
مثال عـملي	43
الحلقة for	44
مثال عـملي	45
الجملة break	46
الجملة continue	48

المعامل الشرطي الثلاثي.. 49
تعرف على المكتبة cmath.. 50

3– **المصفوفات والسلاسل** 53

تعريف المصفوفات 53
الإعلان عـن المصفوفات.. 53
أعضاء المصفوفة 53
الوصول إلى عناصر المصفوفة...................................... 54
مثال عـملي 54
تهيئة المصفوفات 55
أنواع المصفوفات 55
مثال كـودي 55
البحث المتتالي 57
مثال كـودي وحله 57
تصنيف الفقاعات 58
السـلاسـل (المصفوفات الحرفية) 61
إدخال المعلومات في السـلاسـل 61
التابع getline 62
نسـخ السـلاسـل 62
المكتبة ctype 63
بعض دوال الإدخال والإخراج في لغـة السـي القديمة 66
مثال عـملي 68

4– **المؤشرات:** 70

الذاكرة 70
المؤشـرات 72
حجز الذاكرة للمؤشـرات.. 73
الإشـارات أو المرجعيات.. 74
ملاحظات ضرورية حول المرجعيات................................ 75
تحرير الذاكرة 75
فوائد المؤشـرات والمرجعيات 77
مميزات المؤشـرات 77
الميزة الاولى 77
الميزة الثانية 77
الجزء الثالث 79
المؤشـرات الهائمة 79
المؤشـرات الثابتـة 79
المؤشـر void 79
المؤشـرات والمصفوفات.. 80

5– **التوابع:** 81

أسـاسـيات التوابع 81
قواعد مجالات الرؤية 83
المتغيرات الخاصة 83
المتغيرات العامـة 83
المتغيرات السـاكنة 84
مثال عـملي 84
النماذج المصغرة 87
مشـاكل المتغيرات العامـة.. 87

تمرير الوسائط بواسطة القيمة..................................... 88
القيمة العائدة 88
معامل تحديد المدى (::)... 89
الوسائط الافتراضية 89
إعادة أكثر من قيمة بواسطة المؤشرات والمرجعيات........ 90
التمرير بالمرجع أفضل من التمرير بالقيمة...................... 93
التوابع والمصفوفات 93
نقل المصفوفات ذات البعدين إلى التوابع......................... 95
العودية 95
مثال عملي 97
التحميل الزائد للتوابع 99
محاذير عند التحميل الزائد للتوابع.............................. 101
التوابع السطرية 102
تعريف قوالب التوابع 102
كيف يعمل المترجم في حالة القوالب.......................... 104
ماهو القالب 104
زيادة تحميل القوالب 105
ملفات البرمجة (ملفات الرأس)................................... 105
مؤشرات التوابع 108
صفوف التخزين 110
المتغيرات الآلية 110
خلاصة أساسيات وحدة التوابع.................................. 112

6- **مقدمة في البرمجة الكائنية المنحى:** 113

البرمجة الإجرائية 113
البرمجة الهيكلية 113
البرمجة الشيئية 114
مثال: برنامج تسجيل الطلاب في الجامعة........................ 114
إنشاء المثائل (إنشاء كائن) 115
مبادئ البرمجة الكائنية.. 116
الكبسلة أو التغليف 116
الأعضاء ومحددات الوصول... 120
تابع البناء 121
تابع الهدم 123
متى يتم إستدعاء توابع الهدم والبناء 123
التوابع الأعضاء السطرية... 123
المؤشر this 124
الأعضاء الساكنة 125
التوابع الأعضاء الساكنة... 126
الإحتواء أو التركيب 127
اللغة smaltalk والكائنات... 128
لكل كائن واجهة 128
مثال واقعي 129
أمثلة تطبيقية 130
مثال (1) 130
مثال (2) 132
مثال (3) 134

7- **اصنع أنواع البيانات التي تريدها (التحميل الزائد للمعاملات)** 138

مقدمـة في التحميل الزائد للتوابع 138
دوال البناء وزيدة التحميل... 139
تابع بناء النسخـة 142
الخطوة القادمـة 145
كتابة أول معامل للصنف num 145
فائدة للمؤشر this 148
المعامل اللاحق 148
المعاملات الثنائية 151
المعامل (+) 151
معامل الإسـناد 154
تحويل الأنماط 156
عيوب التحميل الزائد 158
المعامل ( ) 158
مثال صنف الأعداد الكسرية Fraction 160

8– **الصنف String :** 167

السـلاسـل في لغـة السـي بلس بلس............................ 167
الإدخال والإخراج مع كائنـات sting 168
إيجاد كلمة ما ضمن سـلسـلة 169
نسخ السـلاسـل 170
التابع ( )substr 170
التابعان ( )begin و ( )end 171
التابع ( )capacity 171
مزيد من التوابع 171
تابع الاستبدال بين سـلسـلتين.................................... 173
تابع المسـح ( )erase 174
حجم الكائن string 175

9 – **الوراثة:** 177

الفرق بين الوراثة في العالم الحقيقي والبرمجي............. 177
مبدأ التجريد 177
الفرق بين الوراثـة والنسـخ أو اللصق 178
اشـتقاق الأصناف 178
دوال الهـدم والبناء 179
مثال على مبدأ الوراثة 180
خلاصة استدعاء دوال البناء عـند التوارث........................ 180
تجاوز دالات الصنف الأب... 183
كيف نستفيد من الوراقة لأقصى حد ممكن...................... 184
طريقة استدعاء الدالة المتجاوزة في الصنف المشـتق 184
الدالات الظاهرية (الإفتراضية) 185
التوارث المتعـدد 188
دوال البناء والهـدم في التوارث المتعـدد......................... 190
الدوال الاخرى وكيفية اسـتدعاؤها 190
الوراثة الظاهرية 192
الأصناف المجردة 194
الدالات الظاهرية الخالصة 194

10– **القوائم المترابطة:** 198

بداية 198

مدخل 198
سلسلة من المؤشرات.. 198
مثال 1 199
عيوب هذه القائمة 205
قوالب الكائنات 206
استخدام القوالب مع القائمة المرتبطة........................... 208
استخدام القوالب مع قائمة أكثر تعقيداً.......................... 211

11– التعامل مع الاستثناءات 214

بداية 214
ما هو الاستثناء؟ 214
التعامل مع الاستثناءات.. 214
مثال عملي 215
كتل catch متعددة 218
مثال عملي 218
الكائنات والاستثناءات 221
الاستفادة من كائنات الاستثناءات............................... 224

12– التعامل مع الملفات 227

بداية 227
العائلة ios 227
الملف Formatted File I / O 227
التعامل مع السلاسل 229
الملفات الثنائية 231
بارامترات الدالة write 232
التعامل مع الأصناف والكائنات 234
التعامل مع الملفات والكائنات بطريقة أكثر تقدماً.............. 236
الدالة open( ) 238
التنقل داخل الملفات 239
كيف تجعل الكائنات أكثر تماسكاً................................ 241
تضمين أوامر التعامل مع الملفات داخل الأصناف 241
الأخطاء عند استعمال الملفات.................................. . 243

13– مكتبة القوالب القياسية 245

بداية 245
محتويات هذه المكتبات.. 245
مقدمة إلى الحاويات 245
كائنات التكرار 245
نظرة عامة إلى الحاويات... 245
المتجهات 247
القوائم 249
الحاوية deque 250
بعض التوابع الأعضاء الآخرين 251
الحاويات الترابطية 252
الحاوية set 253
الخريطة map 255
الخوارزميات 257
خوارزمية البحث 258
خوارزمية الترتيب أو الفرز.. 259

خوارزمية العـد 259
خوارزمية لكل من 260
14 – **مثال عملي** 263

الملاحق:

ملحق (أ) 280
ملحق (ب) 281
ملحق (جـ) 283

بسم الله الرحمن الرحيم

الحمد الله رب العالمين والصلاة والسلام على المبعوث الأمين رحمةً للعالمين محمد ابن عبد الله وعلى آله وصحبه وسلم تسليماً كثيراً ....

فقد أردت حينما ابتدأت فعلياً كتابة هذا الكتاب أن أجعله شاملاً ومجانياً للغة السي بلس بلس، وأنا أقصد بذلك أساسيات السي بلس بلس وليس اللغة بكاملها فهذه اللغة أوسع من أن يضمها ولو مجلد كبير ، فهي واسعة لدرجة لا يكاد يتصورها عقل ، وتتدخل بكافة المجالات في علوم الحاسب وإن شابتها ضعف المقروئية وقلة الإنتاجية ؛ وقد حددت لنفسي شهران ونصف الشهر حتى أنهي ما أعتزمت فعله إلا أني لم أتصور أن يكون تأليف كتاب يتحدث عن أساسيات أي علم سيكون بهذه الصعوبة وبهذا الجهد ، لذلك قلصت فهرس الكتاب ونظمت ما كان في الأمس مسوّدة لكتاب كبير حتى يصبح بهذه الشاكلة التي هي عليه الآن ، وقد بذلك كل جهد وكل غاية حتى ألا يكون في هذا الكتاب خطأ ولو كان غير مقصود ، وإن وقع فهو من نفسي والشيطان وإن لم يكن فهذا بفضل ربي عز وجل .

تزيد صفحات هذا الكتاب عن 270 صفحة ، ولا يتناول هذا الكتاب إلا مبادئ اللغة وأساسياتها وليس مواضيعها المتقدمة أو بالأحرى تخصصاتها البرمجية كبرمجة الشبكات والنظم وغيرها ، ويطيب لي أن أصحبك في نظرة عامة لهذا الكتاب وفهرسه.

في الوحدة الأولى "انطلق مع السي بلس بلس" تناولت فيها أساسيات هذه اللغة وقد عزمت فيها ألا تكون نظرية لدرجة مملة ، كما هو حال أغلب الكتب ، وهذه الوحدة تبدأ فوراً بكود بسيط للغاية ثم يتم شرحه فيما بعد ، وعلى الأقل فهذه طريقة أبجد هوز لتعلم اللغة العربية والتي استخدمها العرب القدامى ، لم أركز في هذه الوحدة على معلومات نظرية تفصيلية مملة بل ركزت على الجانب الكودي وتطبيق الجانب النظري ، فلم أرد الوقوع في عيب الفصل بين النظرية والتطبيق كما هو حال الكثيرين ، وبالرغم من حرصي على ما قلت ، فتعتبر هذه الوحدة أصعب وحدة قمت بتأليفها في الكتاب ، أقصد من ناحية التأليف.

في الوحدة الثانية "بنى التحكم" تعرضت لمواضيع أكثر تقدماً نسبياً بالنسبة للوحدة الأولى وهي بنى التحكم التي تمكنك من كتابة الخوارزميات ، وقد أطلت في كتابة هذه الوحدة لأهميتها وبالرغم من طولها فلم يكن تأليفها صعباً كما هو الحال في الوحدة الأولى ، تتناول هذه الوحدة الحلقات التكرارية for و while .. وغيرها بالإضافة إلى تناولها للمكتبة math .

في الوحدة الثالثة "المصفوفات والسلاسل" تناولت موضوع المصفوفات وبعض تقنياتها ، كيف بإمكانك السيطرة على المصفوفة ، ولم أركز في هذه الوحدة على موضوع المصفوفات بحد ذاتها بل على إعلام القارئ أن هذه المصفوفة مجرد حاوية للبيانات بإمكانك إنشاء ما هو أفضل منها ، وتناولت في نهاية هذه الوحدة موضوع السلاسل في لغة السي القديمة ، نظراً لأن بعض المشاكل لا يتم حلها إلا بها وأيضاً بعض توابع أو دوال العرض.

في الوحدة الرابعة "المؤشرات Pointers" حاولت قدر جهدي ألا تكون هذه الوحدة غامضة كغموض موضوعها ، تعتبر المؤشرات تقنية فعالة للغاية وبداية لك للتفكير كمبرمج حقيقي يسيطر على اللغة وليس كمبرمج تسيطر عليه اللغة ، وكما ترى فإن الوحدات الأربع السابقة تعتبر صغيرة نسبياً وليس كمثل الوحدات القادمة ، قد يشاطرني البعض في تقسيم الكتاب بهذه الطريقة وقد لا يشاطرني الآخرون ، عموماً هذا رأيي وأتمنى أن يكون صحيحاً.

تعرض الوحدة الخامسة موضوع "التوابع Function " حينما تعمل على برنامج كبير نسبياً قد تود تقسيمه إلى أجزاء صغيرة حتى يسهل عليك العمل وأيضاً يفيدك في تصميم البرنامج فكل تابع سيقوم بمهمة بسيطة مما يمكنك من تطوير البرنامج على مراحل وليس على مرحلة واحدة كما هو الحال في الوحدات السابقة ، تتعرض هذه

الوحدة للقوالب التحـميل الزائد والتي هـي أحد التقنيات الجديدة في لغـة السـي بلس بلس عـن القديمـة السـي.

تعرض الوحدة السادسـة موضوع " الكائنـات Object " وهـي فـي الحقيقـة تحـاول إفهام القارئ مبدأ تجريد المعطيات وفائدتـه على مستوى البرمجـة ، في هـذه الوحـدة تبدأ بالسيطرة أكثر فأكثر على اللغـة من خلال مبادئ البرمجـة الشـيئيـة أو الكائنيـة ، ولم أركز في هذه الوحدة إلا على كيفية تصميم الكـائن والأسـاليب الآمنـة ولـو لمحت بشـيء إلى ذلك.

تعرض الوحدة السـابعـة موضوع "التحميل الزائـد للمعـاملات Operator Overloading " حيث يتم تعليمك كيفيـة إنشاء أنـواع جـديدة مـن البيانـات بواسطـة التحميـل الزائـد للمعاملات فبإمكانك صناعـة أنواع خاصـة بك ، وفي نهايـة الوحـدة تعرضنا (أقصـد هـنا المؤلف الذي هـو أنا والقارئ الذي هـو أنت) لمثال بسيط للغاية وهـو عبـارة عـن نـوع جـديد من الأنماط وهـو نمط الاعداد الكسرية Fraction وبالرغم من بدائيـة الصنف إلا أنه يعتبر فرصـة مناسبـة لك للتعرف أكثر على البرمجـة الكائنيـة وإسـتقلالية الصـنف عـمـا سـيؤثر عليه.

تعرض الوحدة الثامنـة موضوع "الصنف string " حيث تجـد الفرق الكبير بين السلاسـل في لغـة السـي ومعالجتها التي تعرضنا لها في الوحدة الثالثة ومعالجـة السلاسـل فـي لغـة السـي بلس بلس ، حيث تناولت الوحدة أغلب مميـزات الصـنف string ، وأتـمنى منك في هذه الوحدة أن تتطلـع أكثـر وأكثـر علـى إمكانـات البرمجـة الكائنيـة وفائـدتها والحلول التي تقدمها والتي تعجـز لغـات البرمجـة الهيكليـة أو تـدفع ثمنـاً غاليـاً للقيـام بنفس العـمليات.

تعرض الوحدة التاسـعـة موضوع "الوراثـة Inheritance " وهـو المبدأ الثاني من مبادئ البرمجـة الكائنيـة ، لم أتعـرض فـي هـذه الوحـدة أو فـي هـذا الكتـاب لموضوع الوراثـة الخاصة ولا سـبب لذلك إلا قصر الوقت في تأليف الكتاب ولم أتعرض بشـكل أكثـر عـمقاً لمبدأ تعـدد الأوجـه فلـم أتنـاول منـه إلا الأسـاسـيات وأيضـاً لـم أتنـاول تـابع النسخـة الظاهري وطريقتـه ، وبالرغـم من هذا القصـور إلا أن هذه الوحدة تعتبر بدايـة جيـدة لـك في مبادئ البرمجـة الكائنيـة.

تعـرض الوحـدة العاشـرة "مقدمــة فـي القـوائم المترابطـة Linked List " وهــو أحـد الخدمات التي تقدمها لغات البرمجـة الكائنية بشكل جيد ، وهذه الوحدة لا تدور إلا فـي مثال واحد يتم شرحـه وتطويره على ثلاث مراحل ، لـم أتعـرض فـي هـذه الوحـدة إلـى بنى معطيات أكثر تقدماً كالأشجار وتعتبر هذه الوحدة بداية جيدة لك للتعامـل مـع بنـى المعطيات.

تعرض الوحدة الحادية عشر موضوع "التعامل مع الاسـتثناءات Handling Exceptions " وتتنـاول هـذه الوحـدة الموضـوع مـن ناحيـة هيكليـة ثـم تتطـور حتـى تـرى كيفيـة اسـتخدامـه من ناحية كائنية أو على مستوى الكائنـات وبالرغـم من ذلك فلا تـزال هـذه الوحدة تقدم لك القليل إذا ما أردت التطور أكثر وأكثر.

تعرض الوحـدة الثانيـة عشـر موضـوع "التعامـل مـع الملفـات Handling With Files " وتتناول هذه الوحدة الموضوع من بدايته حيث تبدأ من تطبيقـه على مستوى التوابع ثم تنتقل إلى متسوى تطبيقـه إلى الكائنـات ، وهذا الأسلوب أفضل فحتى لو كنت مبرمجاً كائنياً بحتاً فقد تحتـاج لتخزين متغيرات في ملفاتك وليس كائنـات ، وبالرغــم مـن تطـور هذه الوحدة إلا أنها لم تتناول كيفية تخزين الكائنات المتوارثـة.

تعرض الوحدة الثالثة عشر موضوع " مكتبة القوالب القياسـية Standard Template Library " وبالرغم من كبر حجم الموضوع وكبر حجم هذه المكتبات إلا أن هذه الوحـدة تحاول أن تبين لك أوجـه الشبـه بين هـذه المكتبـات وكيفيـة أن تسـتفيد منهـا دون أن يكون هـناك أمثلة حقيقيـة في هذه الوحدة.

تعرض الوحدة الرابعـة عشـر موضوع "مثال عـملي" حرصـت فـي هـذه الوحـدة أن يكـون المثـال الـذي سـأتناوله شـاملاً لموضـوع البرمجــة الكائنيــة وقـد تعرضـت مـرة أخـرى لمشـكلة قصر الوقت وقد أردتـه أن يكون مثال آلة حاسـبة كاملـة ، حتـى يفهـم القـارئ

العلاقات بين الكائنات والتصميم الموجه ولغة النمذجة الموحدة UML ، إلا أن رأيي استقر أخيراً ونظراً للمشكلة السابقة على إنشاء حاوية تسلسلية.
أيضاً هناك بعض الملاحق في الكتاب ومنها الملحق "جـ" والذي يعرض لك موضوع المعالج التمهيدي والذي أردته أن يكون وحدة كاملة إلا أن الوقت لم يسعفني سوى أن أجعله ملحقاً بسيطاً في نهاية الكتاب .
هذا الكتاب يركز على البساطة والسهولة وقد حاولت تجنب الشرح الممل الذي لا طائل منه وركزت أكثر على أن تكون المعلومة أكثر تشويقاً دون أن تكون على حساب الناحية العلمية.
ستجد في هذا الكتاب هذه النافذة:

CODE
1. CODE 2. CODE 3. CODE

وهذه النافذة تستثمر لأغراض كتابة الكود.
أرجو من قراء هذا الكتاب إبداء آرائهم أو على الأقل تنبيهي إلى الأخطاء التي ارتكبتها في هذا الكتاب حتى أستفيد منها على الأقل.
أعتذر أيضاً بسبب أخطائي في المصطلحات العربية ، فلقد تعلمت أكثر من نصف ما تعلمته من هذه اللغة بواسطة اللغة الإنجليزية وليس بواسطة اللغة العربية ، وأكثر ما أتخبط فيه من المصطلحات هو مصطلح الـ Function حيث تارةً أرمز له بالتابع وتارةً أخرى أرمز له بالدالة.
بقي أن أشير هنا إلى أنه في حال عدم قدرتك على فتح ملف في برنامج Visual C++ ، فكل ما عليك هو الذهاب النقر على ملف أو file بعد تشغيل البرنامج ثم إلى جديد أو New ثم عبر علامة التبويب Files اختر C++ source file ، ثم أكتب الكود الذي تود كتابته وبعد انتهاءك أنقر على الخيار Build ومنه إلى المترجم compile وبعد أن ينبهك البرنامج إلى أخطائك اضغط على الاختصار Ctrl+F5 حتى يتم تشغيل برنامجك.

سلطان محمد خميس الثبيتي
sultan_altaif@yahoo.com
طالب في جامعة الطائف

الطريق الرئيسي لأساسيات السي بلس بلس

Introduction to C++ Language Programming

الخـطوة الأولـى

سوف تركز هذه الوحدة على إفهامك أساسيات لغة السي بلس بلس ؛ ولتعلم أن أفضل طريقة لتعلم أي لغة برمجية هي البدأ فوراً بكتابة أكوادها ، لذلك ابدأ بكتابة الكود الاول التالي:

CODE

```
1. # include <iostream.h>

2. main()
3. {
4. cout << "Hii C++ " ;
5. return 0;
6. }
```

الكود أعلاه يطبع لك الجملة Hii C++ . **دعنا نقوم الآن بشرح الكود السابق.**

السطر الأول:

هذا السطر يعتبر أحد أهم الأسطر والتي قلما تجد برنامج لا يتضمن مثل هذا السطر . هذا السطر يخبر المترجم بأن يقوم بتضمين المكتبة iostream **في البرنامج ، والمكتبة** iostream **هي التي تقوم بعمليات الإدخال والإخراج في برامج السي بلس بلس؛ حتى تفهم كيف ننطق مثل هذا السطر فإن # تنطق باوند أو هاش وهي تعني موجه ثم كلمة** include **والتي تعني تضمين ثم نلفظ المكتبة** iostream **وهي في الأساس اختصار للجملة** input output stream ، **أي أن السطر الأول يقوم بتوجيه المترجم ليقوم بتضمين المكتبة** iostream **في البرنامج**

السطر الثاني والثالث والسادس:

هذا ما يعرف بالتابع أو الدالة main( ) وجميع البرامج في السـي بلـس بلـس وحتى البرامج المتقدمـة جداً جداً يجب أن تكـون فيها هـذه الدالـة main( ) ، تستطيع أنت أن تقوم بكتابة دوال أخـرى غيـر الـ main( ) لكـن البرنـامج لـن يعـمل إلا بوجود هذه الدالة فهـي اللب الأساسـي لأي برنـامج وكما تلاحـظ فإن الدالة main( ) تبدأ بقـوس فتح في السطر الثالث وتنتهي بقوس إغلاق في السطر السادس ، بينما جميع العبارات والجمل والأوامر التي بـين قـوس الإغلاق والفتح هـي جسـم الدالـة main( ) ، وبـالطبع فلـن يمكـنك أن تقـوم بكتابة أوامر خارج ما يحتـويه هذين القوسـين.

السطر الرابع:

فـي السـطر الأول قمنـا بالطلـب مـن المتـرجم أن يقـوم بتضـمين المكتبـة iostream ، إحدى الخدمات التي تقدمهـا هـذه المكتبـة هـو الكائن cout ، الكـائن cout يخـتص بالمخرجـات ، أي إذا أردت إخـراج أي كتابـات علـى الشاشـة فيجب عليك كتـابة هذه الكلمـة cout بعـد ذلك قمنا بكتـابة حـرفين غـريبين قليلاً ألا وهـما << ، فـي الحقيقـة فهـذين ليسـا حرفـان بـل هـما معامل ، مثلـه مثل عـملية الجمـع أو الطـرح ويسمـى معـامل الإخـراج حيث يقوم بعـمليات الإخـراج أي أن جميـع مـا سـتكتبه لاحقاً سـيقوم الكـائن cout بإخراجـه. بعـد ذلك كتبنا الجملـة المـراد إخراجها ألا وهـي Hii C++ ويجـب عليك أن تنتبه إلى أن الجملة المطبوعـة على الشاشـة بين علامتي تنصيص هـكذا ("Hii C++") بعـد ذلك وضعـنا العلامـة الفاصلـة المنقوطـة ; لنخبر المترجم أن الأمر انتهـى وعليه أن يذهب إلى الأمر التالي.

السطر الخامس:

هذا السطر يجب أن تكتبـه في نهايـة أي دالة سواء أكانـت main أو غيرهـا ، حيث تكتب الكلمـة return 0 ، لن نناقش حالياً مـاذا يعـني هـذا الأمـر ولكـن احرص على كتـابته في أي كـود تكتبــه ، ولاحــظ مـرة أخـرى أن فـي نهايـة الأمر ينتهـي بالعلامـة ; .

ملاحظات ضرورية للغايـة:

هل رأيت الكـود السابق ، تذكر أن أي خـطأ تخـطأ فيـه لن يـتم تنفيـذه ، لـذلك اكتب الكـود كما هـو موضح ولا تحـاول أن تجرب أي أشياء أخـرى.
من أحـد الأخــطاء الشائعــة أن تقـوم بتعـديل السـطر الثالـث وجعـل القـوس هـكذا [، هـذا خـطأ والقوس [يعـني شـيء آخر غير بداية الدالة main( ) .
من أحد الأخـطاء الشائعـة موجودة في السطر الخامس حيث يقوم المبتدئين في البرمجـة بتبديل الـرقم 0 بـالحرف o ، هـذا خــطأ وتـذكر أنــه خــطأ شـنيع للغايـة.
أيضاً أحد الاخـطاء الأخرى والتي قد لا تجـد لها حلاً إذا وقعـت فيها هــو أنــك تقوم بكتابـة أوامـرك بـأحرف كبيـرة هـذا خــطأ ، فـالأمر هـنا لـيس مثـل لغـة البيسك ، فـي لغـة البيسـك لن يهـمك إذا كتبت الاوامر بأحرف صغيرة أو كبيـرة إلا أن الأمر هـنا مختلف فلغـة السـي بلـس بلـس حساسـة لحالـة المحـارف فالكلمـة التي تحتـوي على أحرف كبيرة مختلفـة عـن الكلمــة التـي تحتــوي علـى أحـرف صـغيرة وأغلـب بـرامج السـي بلـس بلـس تحتـوي علـى أحـرف صغيرة وليس أحرف كبيرة ، لذلك تذكر هذا الخــطأ فجميـع مبتـدئي البرمجــة تركـوا البرمجـة من أجل هذا.
قد يصبح الأمر وسواسياً للغاية حينما تقـوم بكتابـة الكــود السـابق فسـوف تتسائل هل أضع مسافـة هـنا هل انتقل إلى سطر جـديد ، لا عليـك مـن هـذا الأمر فبإمكـانك كتابة الكـود السابق ليصبح بهذا الشكل:

CODE

```
1. # include <iostream.h>

2.          main()
3.      {
4.        cout       <<           "Hii C++ " ;
5.                             return 0;}
```

والكودين صحيحان إلا أن الكود السابق أفضل للفهم وأوضح وليس مثل الكود أعلاه ، لذلك احرص على جعل أكوادك منظمة وليست طلاسم سحرية ، ولا توسوس في أمر المسافات البيضاء والعلامات وغيرها.

هذا هو أول مثال كودي احرص على دراسته مرة أخرى إذا لم تفهمه ، صحيح أن الامر صعب في البداية إلا أنه سيصبح متعة كبيرة وخاصة إذا دخلت في مواضيع متقدمة وقمت بكتابة برامج أكثر تطوراً.

الخطوة الثانية

بالنسبة للخطوة الثانية فهذه المرة سنقوم بكتابة كود بسيط ولكنه متقدم بالنسبة لأي مبتدئ برمجة ألا وهو عبارة عن كود يقوم بجمع عددين تقوم أنت بإدخالهما.

CODE

```
1. # include <iostream.h>
2. main()
3.    {
4.    int num1 , num2;
5.    cout << "the first number:\n " ;
6.    cin >> num1;
7.    cout << " the second number:\n";
8.    cin >> num2;
9.    cout << "the Value is: " << num1+num2;
10.   return 0;
11.   }
```

لا مشكلة لديك بالنسبة للأسطر 1 و 2 و 3 و 10 و11 ، إذا لم تفهمها فارجع إلى فقرة الخطوة الأولى.

السطر الرابع:

كما قلنا فالمطلوب أن يقوم مستخدم البرنامج بإدخال عددين اثنين ، ألا تلاحظ معي أن هذان العددان في لغة الرياضيات هما متغيران اثنين ، الامر نفسه بالنسبة للبرمجة فعلينا أولاً اعتبار هذان العددان متغيران وبالتالي نطلب من البرنامج أن يقوم بحجز ذاكرة لعددين اثنين ثم إذا قام

مستخـــدم البرنـــامج بإدخـال عــددين فـإن البرنـــامج يقـوم بأخـــذ العـــددين وتخزينهـما في موقع الذاكرة الذي طلبنا من البرنـامج حجزهـما في البدايــة ، وهذا واضح في السطر الرابع فلقـد قمنـا بتسميــة متغيـران اثنـين الأول هــو num1 والثاني هـو num2 ، الآن كيف يعلم البرنـامج أن num1 و num2 هــما عـددان بإمكانه فعل ذلك عـن طريق أول كلمـة في السـطر الرابـع ألا وهــي int وهـي إختصار للكلمـة integer أي الأعـداد الصحيحـة والاختصـار int هــو عبارة عـن نمط بيانات بإمكـانك عـن طريق تغيير الكلمـة int إلـى char اعتبـار المتغيران num1 و num2 عبارة عـن حرفين اثنين وليس عـددين. لاحـظ أيضاً أن هـناك فاصلـة غيـر منقـوطــة (,) بيـن اسـمي المتغيـران وهـذه ضروريــة فكيف يعرف البرنـامج أنك انتهيت من كتابة اسم المتغير الأول ، ولاحـظ معـي أيضاً أن الأمر انتهى بالفاصلـة المنقـوطـة (;) .
الآن هـناك ملاحـظـة جـديرة بالاهتمـام وهـــي أنــه بإمكــانك تعــديل السـطر الرابع ليصبح سطران اثنين هـكذا:

```
1. int num1 ;
2. int num2;
```

والطريقتين صحيحتان إلا أن الطريقـة الأولـى أفضل بسبب أنها مختصرة.

السطر الخامس والسابع:

السطران الخامس والسابع في أغلبهـما مفهــومان فـلا جــديد فيهــما إذا لـم تفهـمهـما فارجع إلى فقرة الخـطوة الأولـى ؛ إلا أن هـناك أمراً بالغ الأهـميــة؛ لاحـظ معـي الجملة التي طلبنا من البرنـامج طباعـتها:

```
"the first number:\n "
```

كما ترى فإن السبب في أننا طبعـنا هذه الجملة والجملـة في السطر السابع حتى نوضح لمستخـدم البرنـامج أن عليه إدخال العـدد الأول أو العـــدد الثـاني حسب السطر الســابع ؛ ولكـن هـل تـرى آخر الجملـة السابقــة أقصـد هـذه العلامـة ("\n") إن هذه العلامــة لـن يقـوم البرنــامج بطباعــتها بـل إن هـذه العلامـة في الحقيقـة إختصار ، فهـذه العلامـة \n تطلب من مؤشر الكتابـة أن يذهب إلى سـطر جديد وبالتـالي فحينما يقـوم مستخـــدم البرنـــامج بإدخـال العـدد الاول فلن يقوم بإدخاله بجانب الجملة السابقـة بل في السطر التـالي من الجملة السابقـة.
العلامــة \n هـــي تقنيــة فعالـة لتمثيـل المحـارف غيـر المرئيــة أو تلـك التـي تصعب طباعتها فالفعل الـذي تقـوم بــه أشبــه مـا يكـون بالضغــط علـى الـزر ENTER علـى لوحــة المفـاتيح وأنـت فـي محــرر Word أي أن مؤشــر الكتابـة ينتقل إلى سطر جـديد.

السطر السادس والثامن:

بعـكس السطـران الخامس والسابع فإن هـذان السطـران يطلبـان منك إدخـال عـددين اثنـين ، حيـث يقـوم المتـرجم بأخـــذ العـــدد الـذي تقـوم بإدخالــه فـي السطر السادس ويضعـه في المتغير num1 ويأخذ العـدد الذي تقوم بإدخالـه في السطر الثامن ويضعـه في المتغير num2 ، هذه هـي الفكرة ، أما حـــول الكيفيـة فهل تتذكر المكتبـه iostream والكائن cout وما يقومـان بـه ، فـالأمر هـو هـنا نفسـه ، فهـناك كائن جـديد يختص بالإدخال هـو cin وينطـق هـكـذا (سي إن) بعـد ذلك نستخـدم معامل الإدراج وهـو هـكذا >> ولـيس معــامل الإخـــراج الخــاص بالكـــائن cout ، ثــم نكتــب اســم المتغيــر الــذي نريــد مــن المستخـدم أن يقوم بإدخال قيمـة هذا المتغير.

السطر التاسع:

يقوم الكائن cout أيضاً بطباعة المتغيرات ، وفي نهاية الجملة المطبوعة يقوم البرنامج بطباعة هذه العبارة num1+num2 وبما أنها ليست بين علامتي تنصيص فلن يقوم البرنامج بطباعتها كجملة عادية على الشاشة أي هكذا (num1+num2) بل سيقوم بأخذ قيمة المتغير num1 ويجمعها مع قيمة المتغير num2 ويطبع الناتج .
حاول كتابة الكود السابق وتجريبه على جهازك.

الأساسيات

راجع الخطوتان السابقتان وافهمهما جيداً قبل الدخول في هذه الفقرة.

بالرغم من أنك لم تقم إلا بخطوتين فقط في سبيل تعلم لغة البرمجة السي بلس بلس إلا أنها قفزة كبيرة ولا شك وعلى الأقل فقد أعطتك تلك الخطوتان مقدمة عامة عن أساسيات البرمجة؛ فلا بد وأنك صادفت الكلمات التالية:
التعابير ، الأنماط ، المتغيرات ، الكتل ، التوابع ، المكتبات القياسية ، العمليات ، كائنات الإدخال والإخراج.
لا تقلق فبعض الكلمات السابقة لم أذكرها صراحة فيما سبق ولكن تعرضت لفكرتها ، سنبدأ الآن بشرح هذه الأساسيات.أيضاً تتعرض هذه الأساسيات لبعض المواضيع المتقدمة وليس الغرض هو حشو المادة العلمية بل لمعرفة مقدمة ولو بسيطة عنها لأن أصغر كود يحتاج في بعض الأحيان لتلك المعلومات.

المتحولات أو المتغيرات Variable :
المتغيرات كما رأينا عبارة عن أسماء تحجز مواقع في الذاكرة حتى يتمكن البرنامج من تخزين البيانات فيها.
حينما تقوم بتعريف متغير فلا بد أن تخبر المترجم باسم هذا المتغير ونوع المعلومات التي ستحفظها فيه.
حينما تقوم بتحديد نوع المعلومات للمتغير فإن المترجم يحجز له عدداً من البايتات حسب ذلك النوع فمرة تكون بايتاً واحداً ومرة أخرى تكون اثنان ومرة ثمان بايتات.

تسمية المتغيرات:
من الممكن أن يتألف اسم المتغير من أرقام وحروف شريطة أن يكون أول حرف هو حرف عادي وليس رقم ، ولا يسمح بأن يحتوي الاسم على الأحرف اللاتينية أو الرموز مثل ؟ و @ وغيرها ، وتعتبر الشرطة السفلية حرفاً صحيحاً بالإمكان كتابته ضمن اسم المتغير _ ، أيضاً تفرق لغة السي بلس بلس بين المتغيرات ذات الحروف الكبيرة والأخرى ذات الحروف الصغيرة ، وكعادة برمجية جيدة فمن الأفضل أن يكون اسم المتغير اسماً ذا معنى وهذا يسهل عليك الكثير من مهام تطوير الكود وصيانته.

أنماط البيانات وحجومها:
تعرفنا في فقرة الخطوة الثانية على معلومة مهمة للغاية ألا وهي نمط البيانات int ، ولكن لهذا النمط عيب وحيد فهو لا يحتوي على أي علامة عشرية ، وحتى تستطيع من تمكن المتغيرات على التعامل مع الاعداد العشرية فلا بد أن تغير نمط البيانات إلى float ، وإذا أردت أن تغير أيضاً من ذلك لتصبح المتغيرات قادرة على التعامل مع الحروف فلا بد أن تجعل نمطها

هـو char ، بالنسـبة للأعـداد الكبيرة جـداً فبإمكـانك وضـع أنمـاط أخـرى مثـل double و long وجميعها صالحة.

النوع	الحجم	ملاحظات
bool	1	صواب أو خطأ
char	1	0 إلى 256
int	4 وفي بعض الحالات 2	يحوي الأعداد الصحيحة
Float	4	يحوي الأعداد العشرية
double	4	يحوي الأعداد الكبيرة

ملاحظات على الأنماط الرقمية:

بإمكانك استخدام صفات على الأنماط الأساسية ، مثـل الصـفة short و long اللتـان تطبقان على المتغيرات من النوع int :

```
short int number=0;
long int index=0;
```

وبإمكانك إذا ما أردت استخدام هاتين الصـفتين الاسـتغناء نهائياً عــن الكلمـة int ، كما في هذه السطرين:

```
short number=0;
long index=0;
```

الثوابت Constants:

يوجد بعض المتغيرات التي ترغب في عـدم تغييرهـا أبـداً وربمـا حينمـا يصـل البرنامج إلى عـدة آلاف من الأسطر الكودية قد لا تستطيع معرفة إن كان هذا المتغير تغير لذلك فستـود جعله ثابتاً ، وفي حال تغير لأي ظرف مـن الظـروف قـد يكـون خطـأ منـك فسـيقوم المتـرجم بإصـدار خطـأ ينبهـك بـذلك ، وحتـى تستطيع أن تقول للمترجم أن هذا المتغير ثابت ، لذلك لا تسمح لأحد بتغيرهـا حتى أنا المترجم فعليك بكتابة كلمة const قبل نمط المتغير هـكذا:

```
const int number=14 ;
```

تذكر حينما تقوم بالإعلان عـن أن هذا المتغير ثابت فعليك تهيئتـه بقيمــة فـي نفس الوقت وإلا فلن تستطيع تهيئتــه بـأي قيمـة أخـرى لأن المتـرجم يعتبـره ثابتاً ولن يسمح لك بتغييره أي أن السطرين التاليين خاطئين :

```
const int number;
number=14;
```

الإعلانات والتعاريف Declarations and Definitions :

كثيـراً مـا سـتجد فـي هـذا الكتـاب وغيـره مـن كتـب البرمجــة عبـارتي إعـلان وتعريف يجب أن تعرف الفرق بينهـما.

تفرض عليك لغـة السـي بلـس بلـس الإعـلان أو التصـريح عــن المتغيـرات قبـل استخدامها ، أنظر إلى هذا السطر:

```
int number =4;
```

لقد قمت بالإعلان عـن أحد المتغيرات ، أما التعريف فهـو الذي ينشأ عـنه حجز للذاكرة وبالتالي فإن الإعلان السابق هـو نفســه تعريـف لأنـه يصاحبــه حجـز

للذاكرة ، في أغلب المواضيع الإعلان هو نفسه التصريح ولكن تذكر الفرق بينهما لأنه مهم للغاية وخاصة في المواضيع المتقدمة نسبياً كالمؤشرات والكائنات والتوابع وغيرها.

العمليات الحسابية Arithmetic Operations :

في السي بلس بلس توجد خمس عمليات حسابية:

1- عملية الجمع (+) :
2- عملية الطرح (-) :
3- عملية الضرب(*) :
4- عملية القسمة(/) :
5- عملية باقي القسمة (%)

جميع هذه العمليات الحسابية بإمكانك القيام بها على المتغيرات العددية، ولا تقلق فسيأتي الوقت الذي نصل فيه إلى تطبيقها ، بالنسبة إلى العملية الخامسة فلا يمكنك القيام بها إلا على أعداد من النوع int وليس غيره.

عمليات المقارنة أو العلائقية Relation Operator :

في السي بلس بلس توجد عمليات المقارنة حيث بإمكانك مقارنة أعداد مع بعضها البعض أو مقارنة أحرف من النوع char ، وهذه هي عمليات المقارنة في السي بلس بلس:

= =   > =   >   < =   <

لا تقلق فسنصل لفوائد هذه المعاملات في وحدة بنى التحكم مع تطبيقاتها.

التعابير وعمليات الإسناد Assignment Operator And Expressions :

هناك معامل آخر لم نقم بشرحه في العمليات الحسابية وهو المعامل (=) ، هذا المعامل يختلف في السي بلس بلس عن نظيره في الرياضيات، هذا المعامل يقوم بإسناد المتغير الذي في يمينه إلى الذي في يساره وهو يستخدم مع المتغيرات الحرفية فبإمكانك إسناد متغير حرفي إلى آخر ، كما يظهر في هذا المثال:

```
char a=b;
```

في هذا السطر فإنك تخبر المترجم بالقول له أنه يجب عليه أخذ قيمة المتغير b ووضعها في المتغير a .

أيضاً هناك عملية إسناد أخرى ، لنفرض أن لدينا متغير هو i وهو عددي ونريد جمعه بالعدد 2 حينها ستقوم بكتابة:

```
i=i+2;
```

توفر لك السي بلس بلس معامل إسناد أسرع من معامل الإسناد = وأكثر اختصاراً هو += ، بالتالي سنختصر السطر السابق إلى هذا السطر:

```
i+=2 ;
```

التعابير الشرطية Conditional Expressions :

هل تتذكر المعاملات العلائقية ، ستظهر فائدتها هنا لنفرض أن لدينا ثلاثة متغيرات ، حيث أننا نقوم بكتابة برنامج يقوم بمقارنة أي عددين وحساب الأكبر منهما ، لنفرض أن المتغيرين أو العددين الذي نود مقارنتهما هما a و b ، أما المتغير الثالث فسيكون max .

```
1     if (a > b )
2           max = a ;
3     if ( b < a)
```

```
4          max = b ;
5     if ( b == a)
6          max = a = b;
```

هنا أحد التعابير الشرطية وهو التعبير if يقوم هذا التعبير باختبار التعبير الذي بين القوسين بعده ، وفي حال نجاح التعبير فإنه ينفذ الأوامر التي بعده وفي حال عدم نجاحه فإنه يخرج تلقائياً ولا ينفذ الأوامر التي ضمن الكلمة if .
انظر إلى السطر الأول ، لنفرض أن المتغير a بالفعل هو أكبر من المتغير b حينها سيتم تنفيذ السطر الثاني أما في حال لم يكن كذلك فلن يتم تنفيذ السطر الثاني وسيواصل البرنامج عمله وينتقل إلى السطر الثالث.
انظر أيضاً إلى عملية المقارنة في السطر الخامس وهي == أي هل يساوي المتغير a المتغير b ، في حال كانا متساويان فإن السطر السادس سيتم تنفيذه ، انظر أيضاً أننا في حالة المساواة لم نقم بكتابة المعامل = ، والسبب أن المعامل = كما قلنا سابقاً هو معامل إسناد أي يأخذ القيمة التي على يمينه ويضعها على يساره ولا يقوم بمقارنة أبداً أما المعامل == فيقارن بين القيمتين .

عمليات الإنقاص والإزادة Increment and Decrement Operators :

سنتعرف الآن على عملية غريبة علينا وهذه العمليتين هي عملية الإزادة ++ وعملية الإنقاص -- .
ليس ذلك فحسب بل طريقة كتابة هذه العمليتين قد تختلف ، وهي صيغتين إما أن تكون إحدى هذه العمليتين على يمين المتغير وإما على يساره وتختلف في كلا الحالتين ، حتى تفهم ما أعنيه لنفرض أن لدي متغيران الأول هو a والثاني هو b ، انظر إلى هذه الأسطر:

```
a = ++b ;
```

إن هذا السطر يخبر المترجم بالقول يا أيها المترجم زد قيمة المتغير b رقماً واحداً (أي العدد 1) ثم أسند قيمة المتغير b إلى المتغير a. فلو افترضنا أن قيمة المتغير b هي 6 ، فحينما يقوم البرنامج بتنفيذ السطر السابق فإنه يقوم أولاً بزيادة المتغير b زيادة واحدة أي تصبح قيمته 7 ثم يسند القيمة إلى المتغير a ، أي ستصبح قيمة المتغير a أيضاً 7 ؛ الآن لو افترضنا أننا قمنا بكتابة صيغة أخرى وهي هكذا:

```
a = b ++ ;
```

ستختلف العملية هنا ، والآن قم بالتركيز فيما سيكتب ، أولاً سيأخذ المترجم قيمة المتغير b بدون أي تغيير ويقوم بإسنادها إلى المتغير a ثم بعد ذلك يقوم بزيادة المتغير b زيادة واحدة ، أي أن هذه الصيغة عكس الصيغة السابقة فلو فرضنا أن قيمة المتغير b هي 6 ، فأولاً سيأخذ المتغير هذه القيمة ويسندها إلى المتغير a ، وبالتالي تصبح قيمة المتغير a هي 6 ثم بعد ذلك يقوم المترجم بزيادة المتغير b ، أي أنها ستصبح 7 .
أتمنى أن تكون الصيغتان مفهومتان ، أيضاً نفس الشرح السابق يطبق على عملية الإنقاص -- ، مع إختلاف العمل الذي تقومان به طبعاً.

المعامل sizeof :

هناك معامل آخر وهو المعامل sizeof ، حيث أن هذا المعامل يحسب لك حجم المتغيرات أو أي شيء آخر ومن الممكن استخدامه بهذا الشكل:

```
sizeof (int) ;
```

حيث يحسبك لك حجم نمط البيانات من النـوع int ، أما إذا أردت حسـاب أحـد المتغيرات فبإمكانك استخدامه بدون أقواس ، أي هـكذا:

```
sizeof a ;
```

حيث a متغير .

القراءة (الإدخال) والكتابة:

بإمكانك الطلب من البرنامج طبع أي قيمة علـى الشاشــة بواسـطـة الكـائن cout ، وبإمكــان هــذا الكــائن طباعــة أي قيمــة عبـر معامـل الإخـراج >> ، وبإمكانـه طباعـة المتغيـرات أو الجمـل التـي أنـت تريـد إظهارهـا ولكـي تظهـر جمـل علـى الشاشــة فعليـك كتابتهـا بـين علامتـي تنصيص ، كمـا فـي هـذا المثال:

```
cout << "Hellow C++";
```

أما إذا أردت إظهـار قـيم أحـد المتغيـرات فعليـك كتابـة اسمــه دون علامتـي تنصيص كما هـنا:

```
cout << a ;
```

مع العلم أن a عبارة عـن متغير.

أيضاً فبإمكانك طباعـة أكثر مـن متغير أو جملة دفعـة واحـدة ،كمـا فـي هـذا السطـر:

```
cout << "Please: " << a << b << "Hellow" ;
```

أيضاً هـناك عبارة بإمكانك إستخدامها لإفراغ المنطقة الوسيطـة مـن جميـع الاحرف العالقـة أو بشكل مبتدئ طباعـة سطر جـديد ، انظر إلى هذا السطـر:

```
cout << "Hellow" << endl << "World" ;
```

سيكون مخرج هذا الأمر على الشاشـة هـكذا:

```
Hellow
World
```

أيضـاً هــناك بعـض الخصـائص للكـائن cout وهـي سلاسـل الإفـلات ، وقـد استخدمنا أحدها في المثالين السابقين وهـو \n والذي يقوم بطباعــة سـطـر جـديد لك.

بعض سلاسل الإفلات:

\t	جدولة أفقية تترك 3 فراغات.
\n	الإنتقال إلى صفحـة جديدة.
\r	إعادة المؤشر إلى بداية السطـر.
\a	يقوم بإصدار صوت تنبيه.
\b	الحذف الخلفي (back space).

سلاسل الإفلات نقوم بكتابتها ضمن الجمل أي بين علامتي التنصيص " " .

بالنسبة للإدخال في السـي بلس بلس فبامكانك بواسطـة الكائن cin ، وهـذا الكائن يستخدم فقط مع المتغيرات ولـيس شــيء أخـر ، وقـد رأيـت بعضاً مـن استخداماته في المثالين السابقين

مساحات الأسماء:

جميـع المتغيـرات لهـا اسـم ولـيس ذلـك فحسـب بـل تقريبـاً كـل شــيء فـي البرنامج له اسم ، وحينما تقوم مثلاً في المستقبل بكتابـة بـرامج كبيـرة مثـل الوورد أو أنظمـة تشغيل وغيرها فحينها ستقوم بتسمية الكثير من المتغيـرات

والتوابع والكائنـات ، هذه الكائنـات والتوابع والمتغيرات قـد تشـترك فـي اسـم ما وسيكون من المتعب لك تغيير مسمـى أحـد هـذه الأشـياء لأنـك إن غيرتـه فستقوم بتغيير اسمـه في كل الأماكن التي ذكرت.
ظهرت قريباً للسـي بلس بلس تقنية جـديدة وهـي مساحات الأسماء ، وهـي تقوم بتغليف المتغيرات والتوابع والكائنات باسـم معـين ، أيضـاً حينمـا تقـوم بكتابة مكتبـة لك فعليك بتغليفها بمساحـة أسماء ، لـن نناقش هـنا موضـوع مساحـات الأسماء ، ولكن عليك تـذكر أن مكتبـة iostream تسـتخدم مساحـة الأسماء std ، وتعلم أنت أنـك تسـتخدم الكائنـات cin و cout التابعـات للمكتبة iostream ، لذلك فعليك أيضاً أنت استخدام نفس مساحـة الأسماء ووسيلتك إلى ذلك هـو كتابة هـذا السـطر فـي أعلـى البرنـامج بعـد أن تقـوم بتضـمين المكتبات فوراً.

```
using namespace std;
```

ومعنـى ذلـك أنـك تخبـر المتـرجم إذا وجـدت أي شـيء لا تعـرف لـه مساحـة أسماء فكل ما عليك هـو إفتراض أن مساحـة الأسماء الخاصة به هـي std .
لا تقلق فسنتعرض لجميع هذه المسائل في وقت لاحق ، احرص على فهـم ما تم ذكره ولا شـيء آخر.

التعليقات:
حينمـا يصـبح برنامجـك كبيـراً للغايـة فعليـك دائمـاً اسـتخدام التعليقـات ، لا تستخدم التعليقـات فـي جميـع أسـطر برنـامج بـل فقـط فـي المواضـع التـي تعـتقد أن هـناك صعـوبة في فهـمها حينمـا سـيأتي غيـرك لقراءتها أو حينمـا تأتي أنت بعـد مضـي مدة طـويلة لتقرأ تلك الأكـواد.
حينما تقـوم بكتابة تعليق فعليك إخبار المترجم ألا يقوم بقراءة هـذا التعليـق ، ووسيلتك إلى هذه هـي العلامـة // ، انظر إلى هذا السطر:

```
int a=0   // this is a
```

تذكر حينما تقـوم بكتابة هذه العلامـة // فإن المترجم لن يقوم بقراءتها أبداً أو بقراءة الكلمات التي ستقع بعـدها ضمن نفس السطر الموجودة فيه ، أما لـو كتبت أي شـيء آخر بعـد السطر كتعليق فسـيقوم المتـرجم بقراءتـه وإصـدار خطأ بذلك
هـناك علامـة تعليق أفضل أخذتها لغـة السـي بلـس بلـس مـن لغـة السـي وهـي علامـة /* ، حينما تكتب هذه العلامـة فلن يقرأ المترجم ما بعـدها ليس من نفس السطر بل كل ما في الكـود حتى تكتب هذه العلامـة */ ، انظر إلى هذا المثال:

```
int a=0   /* the compiler
cannot read thie*/
```

هذا هـو تقريباً أهـم ما تحتاجـه في أساسيات السي بلـس بلـس والآن إلـى قليل من الأمثلة حتى تفهـم ما تم كتابته سابقاً.

مثال (1)
قم بكتابة كـود يقوم بعرض الجملة التالية على الشاشـة.

```
Hellow Worlad
I am a programmer
```

الحل:

كما تـرى فإننا هــنا لـن نســتخدم أي متغيـرات (تـذكر: المتغيـرات تسـتخدم لتخزين ما نريد تخـزينـه في الذاكرة) لأننا لن نقوم بتخـزين أي شــيء بـل كـل ما علينا فعله هـو عرض بعض الجمل على الشاشـة ، الآن إلى الكـود:

CODE

```
1. #include <iostream>
2. using namespace std;

3. int main()
4. {
5. cout << "Hellow World\n I am a programmer " << endl;
6. return 0;
7. }
```

كما ترى فلم نستخدم إلا سطراً وحيداً لتنفيذ المطلوب من السـؤال أو المثـال وهـو السطر الخامس ، انظر في السـطر الخـامس إلـى سلسـلة الإفـلات \n كما قلنا تستخدم هذه السلسلة للإنتقال إلى سطر جـديد.
انظر أيضاً إلى السطر الأول ، انظر إلى الاخـتلاف بينــه وبـين الأسـطر الأولـى في الأمثلة السـابقـة تجـد أننا لم نقوم بكتابة الإمتداد (.h) والسبب فـي ذلـك هـو وجـود السـطر الثـاني الـذي كمـا قلنا يسـتخدم مساحـة الأسـماء std ، وهـناك أسباب أخرى لكن لن نذكرها لأنها من المواضيع المتقدمـة جداً لـذوي البرمجـة المبتدئين ، حاول دائماً وأبـداً أن تسـتخدم نفس نسـق هـذا المثـال وليس الامثلة السـابقـة.

مثال (2):
قم بكتابة كـود يتأكد إن كان العـدد الذي سيدخله المستخدم هـو عـدداً فردي أو زوجي.

الحل:
أولاً كما ترى فإن هذا البرنامج يقوم بعـملية اتخاذ قرار ألا وهـو إن كان العــدد فردياً أو زوجياً ، لذلك علينا استخدام العبارة if الشرطيـة.
الآن علينا التفكير كيف سنجعل البرنامج يقرر إن كان العـدد المدخل زوجياً أم فردياً ، وسيلتنا الوحيد لذلك كما تعلم أن العـدد الزوجي يقبل القسمــة علـى 2 أما العـدد الفردي فلا يقبل القسمـة على 2 ، أي أن خارج القسمــة للعــدد الزوجي على 2 هـو 0 ، أما إن لم يكن خارج القسمــة عليه هــو 0 فسـيكون عـدداً فردياً بالتأكيد.
هـناك قضية ثانية وهـي كيفية إعلام المستخدم بأن العـدد زوجـي أو فـردي ووسيلتنا إلى ذلك هـي كتابة عبارة على الشاشــة تخبره بذلك.
كما ترى فإن هـناك عـدداً مدخلاً وبالتالي فسنستخدم الكـائن cin وكما تـرى فإن الكائن cin يجب أن يكون هـناك متغيرات لاستخدامـه ، انظر إلى الكـود:

CODE

```
1. #include <iostream>
2. using namespace std;
3. int main()
```

```
4. {
5. int a=0;
6. cout << "Enter The Number:\t";
7. cin >> a;
8. if (a%2==0)
9. cout << "\nThe Number is divide by 2\n"
10.   return 0;
11.   }
```

لاحظ هنا أن هذا البرنامج قام بالإعلان عن متغير من النوع int وستعرف لماذا ثم طلب من المستخدم إدخال رقم لاختباره في السطر 7 ، في السطر 8 يقوم البرنامج بقسمة العدد المدخل على 2 وإذا كان باقي القسمة يساوي 0 فسيقوم بتنفيذ السطر 9 أي طباعة أن هذا العدد زوجي ، أما إذا لم يكن كذلك فلم يقوم البرنامج بأي شيء.
ستقوم أنت بتطوير المثال السابق حتى يقوم بعمليات أكثر تعقيداً حينما تفهم محتويات الوحدة الثانية.

الثوابت الرقمية:

هناك أيضاً بعض التقنيات في السي بلس بلس وهي الثوابت المرقمة .
لنفرض أنك تقوم بكتابة كود للتواريخ وأنك تود إنشاء سبع متغيرات كل متغير يحمل اسم يوم من أيام الأسبوع.
توفر لك لغة السي بلس بلس آلية مميزة لاختصار الكود والوقت والجهد وهي الثوابت الرقمية ، سنقوم الآن بكتابة سطر يحوي ثلاثة أيام من الأسبوع فقط.

```
      enum Days { sat , sun , mon };
```

كما ترى فلقد استخدمنا الكلمة المحجوزة enum والتي تعني الإعلان عن قائمة ثوابت مرقمة أما الكلمة Days فهي المسمى.
الآن لنفرض أننا لم نقم باستخدام هذه التقنية أو لنتساءل كيف سيقوم المترجم بترجمة السطر السابق ، أنظر إلى الأسطر التالية:

```
      const int sat = 0;
      const int san = 1;
      const int mon = 2;
```

كما ترى يبدأ المترجم العد من الصفر ، وأنت لا تريد فعل ذلك لأنه لا وجود لتاريخ 0 ، لذلك بإمكانك إعادة كتابة السطر السابق كما يلي حتى تحل هذه الإشكالية:

```
      enum Days { sat = 1 , sun , mon } ;
```

سيقوم البرنامج الآن بالعد من الرقم 1 وليس الصفر.
لم يذكر هذا الكتاب الكثير من الأمثلة حول الثوابت المرقمة وليس السبب في قلة استخدامها بل إلى تقصير من نفسي وأعتذر عن هذا.

التوابع function( ) :

سنتعرض للتوابع في وحدة لاحقة ولكن يجب عليك أن تفهم ولو مقدمة بسيطة بشأن هذا الموضوع.

يتألف أصغر برنامج من تابع واحد على أقل تقدير ألا وهو التابع main ( ) ، **والذي رأيناه في الأمثلة السابقة .**
لا يمكنك تضمين أي أوامر خارج تابع ما فالتوابع عبارة عن كتل تقوم بضم الاوامر والمتغيرات في كتلة واحدة وهي تقوم بعمل ما ثم ينتهي عملها وقد تقوم بإسناد المهمة إلى تابع آخر أو لا تقوم بأي شيء أصلاً في بعض الحالات.
عموماً الهدف من هذه الوحدة هو إعطاؤك لمحة أساسية عن البرمجة أو اللغة سي بلس بلس بمعنى أصح ، من الضروري أن تفهم الخطوتين الأولى والثانية إن لم تكن ملماً بأساسيات ، أما بقية الوحدة فلا يفترض منك أن تلمها حالاً بل فقط أن تأخذ لمحة عنها لأن أغلب المواضيع اللاحقة ستتناول جميع الذي ذكرناه بالشرح والتفصيل الذي أرجو ألا يكون مملاً.



Control Flow

بدايـة:

لقد أنجزنا بعضاً من الأكـواد المفيدة بواسـطـة القليل من المعرفـة في اللغــة ؛ إلا أن الأمر لن يستمر مطولاً هـكذا ، فماذا لو طـلـب منـك إنشــاء برنـامج آلـة حاسبـة متكاملـة تقوم بجميع العـمليات وليس بعـملية واحدة ، أيضـاً مـاذا لـو طلب منك كتابة برنامج يطلب من المستخدم إدخال قيم أكثـر مـن 100 متغيـر للقيـام بعــمليات حسابيــة أو لكتابـة قاعــدة بيانـات ، حينهـا ســيزداد الكـــود لدرجـة مملة للغايـة ، من هـنا تظهـر فائـدة بنـى التحـــكم، والتـي تسـمـح لـك بالتحكم أكثر في برنامجك.

جمل بنى التحكم:

تقسم جمل بنى التحكم إلى قسمين رئيسيين ؛ هـما:

1- **جمل إتخاذ القرارات.**
2- **جمل تنفيذ الحلقات.**

وسنتعرض لكلا النـوعين بالشرح والتفصيل.

جمل إتخاذ القرار:

تفيد جمل اتخاذ القرار كثيراً في الاكـــواد ، فهـــي تســمـح لـك بالســيطرة أكثـر على برنامجك ، أيضاً فلو ألقينا نظرة متفحصـة للأكـواد السابقـة فستجد أنــه لا يمكنك السماح للمستخدم بالتفاعل مع البرنـامج ، انظر إلى برنـامج الـوورد ، إنـه يعـطيك خيارات واسعـة مـن خـلال شــريط الأدوات ولـيس مثـل البـرامج التي نكتبها حالياً ، من هـنا تكمن أهـمية وفائدة جمل اتخاذ القـرار ، وتـذكر أن هـناك جملتين رئيسيتين ؛ هـما:

1- **الجمـلة if وتفرعاتها.**
2- **الجملة switch .**

الجمـلة if:

تأخذ الجملـة if الصيغـة العامـة التالية:

```
if (expression) {
            statement1;
            statment2;
              }
```

بإمكاننا الإختصار إلــى القــول أنــه إذا كــان الشــرط الــذي تقـوم الجملــة (if) بإختباره صحيحاً فقم بتنفيذ الجمل التي بين القوسين وفي حال عـدم صحــة الإختبار فلا تقم بتنفيذ الجملـة if وإنمـا اســتمر فـي قـراءة البرنـامج مـن بعــد كتلة if .

فمثلاً انظر إلى هذا الكـود:

CODE

```
1- #include <iostream>
```

```
2- using namespace std;

3- int main()
4- {
5- int i=0 ,j=0;
6- cin >> i >> j ;
7- if (i > j )  {
8- cout << "The number i is bigger than j" ;
9- }
10-       return 0;
11-       }
```

كما ترى فإن هذا الكود يطلب من المستخدم إدخال رقمين ، يقوم البرنامج بمقارنة هذين الرقمين وفي حال إذا كان الرقم الأول أكبر من الرقم الثاني فإنه يطبع رسالة تخبرك بذلك وفي حال أن العددين متساويين أو أن العدد الثاني هو أكبر فلن يتم تنفيذ السطر 8 لعدم صحة شرط الجملة if .

الجملة if/else:

لا يقوم الكود السابق بفعل أي شيء إذا اختل شرط الجملة if وبالرغم من أنه بإمكاننا كتابة جملة if ثانية في حال مساواة العددين وجملة if ثالثة في حال أن العدد الثاني أكبر ، إلا أن ذلك لا يمنع من وقوع أخطاء ، فمثلاً فإن بعض الأشخاص لن يتوقعوا أبداً أن العددين سيكونان متساويان لذلك فإن الحل الأفضل هو أن يكون هناك جملة أخرى موازية للجملة if تبدأ في العمل في حال عدم نجاح إختبار الشرط في الجملة if .
الصيغة العامة لهذه الجملة هي كالتالي:

```
if (expression) {
                statement1 ;
                statement2;
               }
else      {
           statement3;
           statement4;
         }
```

بإمكاننا إختصار هذه الجملة إلى القول: أنه في حال عدم نجاح إختبار الشرط في الجملة if فإن البرنامج سيقوم بتنفيذ الكتلة التي تتبع للعبارة else ، أما في حال نجاح اختبار الشرط في الجملة if فإن البرنامج سيقوم بتنفيذ الكتلة التي تتبع للجملة if ولكنه سيتجاهل الكتلة التي تتبع الجملة else .

الآن سنقوم بإعادة كتابة الكود السابق وهذه المرة سنجعله يتعامل مع الحالات الأخرى.

CODE

```
12-      #include <iostream>
13-      using namespace std;

14-      int main()
15-      {
16-      int i=0 ,j=0;
17-      cin >> i >> j ;
18-      if (i > j )  {
19-      cout << "The number i is bigger than j" ;
20-      }
21-      else { cout << "error" ; }
22-      return 0;
23-      }
```

لم يختلف الكود الحالي عن الكود السابق إلا في السطر 21 حينما جعلنا البرنامج يعرض على الشاشة رسالة خطأ للمستخدم في حالة عدم نجاح اختبار الشرط في العبارة if .

العبارة if/else من الممكن أن نطلق عليها العبارة if الثنائية الإتجاه لأن البرنامج يتفرع فيها إلى طريقين أو إلى فرعين بعكس الجملة if السابقة فإنها توصف بأنها أحادية الإتجاه لأنها تسلك طريقاً واحداً في حال نجاح الشرط.

بقي أن نشير هنا إلى ملاحظة ضرورية هامة ، جميع جمل بنى التحكم بما فيها العبارتين السابقتين لا تنفيذ في حال نجاح الشرط إلا عبارة واحدة فقط ، أما في حال إذا أردت أن تقوم بتنفيذ أكثر من عبارة أو سطر برمجي فعليك كتابة هذه الجمل في كتلة واحدة بين قوسين كبيرين اثنين.

الجملة else/if:

من الممكن وصف هذه الجملة بأنها متعددة الإتجاهات ، فهي تسمح لك بسلوك الكثير من الطرق بدلاً من طريق واحد فحسب ، انظر إلى صيغتها العامة.

```
if (expression) {
            statement1;
            statement2;
            statement3;
             }
else if (expression) {
           statment1;
```

```
                }
else if (expression) {
          statement;
                  }
else        {
            statement;
          }
```

سنقوم الآن بتطوير الكود السابق ليصبح قادراً على التعامل مع جميع الحالات.

CODE

```
1- #include <iostream>
2- using namespace std;

3- int main()
4- {
5- int i=0 ,j=0;
6- cin >> i >> j ;
7- if (i > j )  {
8- cout << "The number i is bigger than j" ;
9- }
10-       else if (j > i)  {
11-       cout << "The number j is bigger than i" ;
12-       }
13-       else if ( j=i) {
14-       cout << "there is no bigger number" ;
15-       else   { cout << "error" ; }
16-       return 0;
17-       }
```

ترى الإختلاف عن الأكود السابقة في هذا الكود في الأسطر 10 إلى 15 وقد أضفنا لهذ الكود جملتين else if ، تقوم الأولى بإختبار ما إذا كان العدد الثاني هو الأكبر ثم تطبع جملة تخبر المستخدم بذلك أما الثانية فهي تقوم بإختبار ما إذا كان العددان متساويان وتطبع جملة تخبر المستخدم بأنه ليس هناك رقم أكبر من الآخر أما العبارة else الأخيرة فهي تفيدك في حال وقوع مفاجآت جديدة.

قد تتحاذق وتتساءل عن الفائدة المرجوة من العبارة else/if وقد تقوم بتعديل الأسطر 30-38 إلى الشكل التالي:

```
1- if (i > j )  {
2- cout << "The number i is bigger than j" ;
```

```
3- }
4- if (j > i)  {
5- cout << "The number j is bigger than i" ;
6- }
7- if ( j=i) {
8- cout << "there is no bigger number" ;
9- else   { cout << "error" ; }
```

أي أنك ستقوم بالاستغناء عن العبارة else/if بالعبارة if ، ولهذا الرأي عيوب كثيرة وأخرى شنيعة قد تدمر برنامجك وقد تجعله مضحكاً.

1- في حال إستخدامنا للعبارة else/if فإنه في حالة نجاح أي شرط من شروط العبارة else/if فإن البرنامج يخرج من هذا التداخل الحاصل من عبارة else/if ، ولن يقوم بإجراء أي إختبار وهذا له فائدة كبيرة فهو يقلل من المجهود الذي يقوم به الحاسب وبالتالي يحسن نواحي كثيرة من برنامجك ، أما في حال إستخدام العبارة if فإنه حتى في حال نجاح أي شرط من شروط العبارة if فإن البرنامج سيستمر في إختبار الرقم حتى يخرج نهائياً، وبالتالي فهذا يزيد من المجهود الملقى على الحاسب مع العلم أن هذا المجهود سيفيدك كثيراً إذا ما كنت تعمل على مشروعات كبيرة وليس مثل هذا الكود الصغير.

2- إذا كنت حذقاً للغاية فستجد أن الكود الذي يستخدم العبارة if ، لم يستخدمها هي لوحدها بل إستخدم أيضاً العبارة else ، وهذه العبارة ليس لها أي علاقة بالعبارتين if السابقتين ، فلنفرض أن العدد i أكبر من العدد j ، فإن الشرط في السطر الأول سينجح ويقوم البرنامج بكتابة جملة تخبر المستخدم بذلك وسينتقل التنفيذ إلى السطر 4 ولن ينجح اختبار الشرط بالطبع وبالتالي سيتجاهل البرنامج السطرين 5 و 6 وينتقل إلى اختبار الشرط في السطر 7 والذي لن ينجح بالتأكيد ، الآن سينتقل التنفيذ مباشرة إلى العبارة else في السطر 9 لأن إختبار الجملة if في السطر 7 لم ينجح وكما تعلم فإن العبارتين مرتبطتين ببعضهما وليس لهما أي علاقة بالجمل if السابقة وبالتالي فسيقوم البرنامج بطباعة رسالة خطأ حسب السطر 9 ، وستجد أن برنامج أصبح مضحكاً ، أما لو قمت باستخدام العبارة else/if فإن أياً من ذلك لم يكن ليحدث.

ملاحظة: يعتبر الخطأ السابق أحد أصعب الأخطاء والذي قد تحتار فيه لدرجة تجعلك تترك الكود الذي تعمل عليه لذلك احرص على البرمجة الآمنة وليس البرمجة الخطرة.

مثال عملي:

سنقوم الآن بكتابة برنامج شبيه ببرنامج الآلة الحاسبة.

CODE

```
1- #include <iostream>
2- using namespace std;
3-
4- int main()
5- {
```

```
6-    float a,b;
7-    char x;
8-
9-    cout << "Enter Number1:\t" ;
10-         cin >> a;
11-         cout << "Enter Number2:\t" ;
12-         cin >> b;
13-         cout << "Enter the operator\t";
14-         cin >> x;
15-
16-         cout << endl << endl;
17-
18-         cout << "Result:\t";
19-
20-
21-         if (x=='+') { cout << a+b ;}
22-         else if (x=='-') { cout << a-b;}
23-         else if (x=='*') { cout << a*b;}
24-         else if (x=='/') { cout << a/b;}
25-         else            { cout << "Bad Command";}
26-
27-         cout << endl;
28-
29-         return 0;
30-   }
```

يطلب البرنامج من المستخدم إدخال العدد الأول ثم العدد الثاني ثم العملية الحسابية التي تريد استخدامها تبدأ عبارات الجملة else/if من السطر 21 وتستمر حتى السطر 25 ، حيث تختبر المتغير x لترى إن كان يقوم يحتوي على أي من العمليات الحسابية وفي حال ذلك فإنها تطبع القيمة الناتجة وفي حال عدم ذلك فإن التنفيذ سيكون في السطر 25 حيث في حال أدخل المستخدم أي عملية أو حرف أو حتى رقم من غير العمليات الحسابية المعروفة فإن البرنامج يطبع عبارة تنبئك بحدوث خطأ ، بعد ذلك يخرج البرنامج نهائياً.

لاحظ أنه إذا كان الحرف x عبارة عن عملية حسابية فإن الجملة else/if لن تقوم بإجراء العملية الحسابية وتخزينها في متغير بل ستقوم بطباعة القيمة فوراً وإجراء العملية الحسابية عليها في نفس الوقت.

الجملة switch:

الجملة switch إحدى جمل إتخاذ القرارات ، إلا أنها هذه المرة تعتبر جملة if متطورة ، حيث أنه ليس هناك أي فرق بينها وبينها الجملة if متعددة الإتجاهات ، وتصاغ هذه العملية حسب الصيغة التالية:

```
switch (expression) {
        case const-expr: statements ;
        case const-expr: statements ;
        default:  statements ;
         }
```

بإمكاننا إختصار شرح هذه الصيغة العامة ، إلى أنه بإمكانك أن تكتب المتغير الذي تريد إختباره (في مثال الآلة الحاسبة كان المتغير x) وتكتبه بين قوسين بعد عبارة switch ، بعد ذلك تقوم بكتابة الحالات المتوقعة لهذا المتغير بعد الكلمة الدليلية x ، وفي حال مطابقة إحدى هذه الحالات مع المتغير يتم تنفيذ الجمل التي تختص بتلك الحالة وفي حال عدم موافقة أي منها فبإمكانك كتابة حالة عامة (تشبه الجملة else في مثال الآلة الحاسبة) ، قد ترى أن هناك الكثير من التشابه بين الجملة else/if والجملة switch ، إلا أن هناك بعض الفروق البسيطة التي قد تكون مؤثرة في بعض الأحايين :

1- في حال مطابقة إحدى الحالات مع المتغير المراد اختباره فإن الحالة نفسها لا تُعتبر خيار من خيارات متعددة بل تعتبر نقطة بداية لتنفيذ العبارة switch ؛ بالنسبة للجملة else/if فإن الأمر يعتبر خيارات وليس نقطة بداية.
2- في حال تنفيذ إحدى الحالات فإن البرنامج لا يخرج من الجملة switch بل يستمر في التنفيذ والبحث عن حالات أخرى مشابهة وفي حال وجدها يقوم بتنفيذها ، بإمكانك الخروج من الجملة switch إذا أردت عبر الكلمة الدليلية break ، وفي حال عدم رغبتك في الخروج فإن البرنامج سيستمر في البحث عن حالات مشابهة حتى يصل للحالة العامة default ويقوم بتنفيذها على الرغم من وجود حالات أخرى مطابقة.

الآن سنقوم بإعادة كتابة مثال الآلة الحاسبة ، ولكن هذه المرة بالعبارة switch وسترى الفرق بينها وبين الجملة else/if :

CODE

```
1-   #include <iostream>
2- using namespace std;
3-
4- int main()
5- {
6-    float a,b;
7-    char x;
8-
9-    cout << "Enter Number1:\t" ;
```

```
10-         cin >> a;
11-         cout << "Enter Number2:\t" ;
12-         cin >> b;
13-         cout << "Enter the operator\t";
14-         cin >> x;
15-
16-         cout << endl << endl;
17-
18-         cout << "Result:\t";
19-
20-         switch (x) {
21-         case '+':
22-               cout << a+b ;
23-               break;
24-         case '-':
25-               cout << a-b;
26-               break;
27-         case '*':
28-               cout << a*b;
29-               break;
30-         case'/':
31-               cout << a/b;
32-               break;
33-         default:
34-               cout << "Bad Command";
35-         }
36-
37-         cout << endl;
38-
39-         return 0;
40-   }
```

هذا هـو البرنامج بشكل عام وبالنظر إلى أني قمت بشرحـه سـابقاً فسـأقوم بشرح عبارة switch فحسب ، انظر:

```
1- switch (x) {
2-    case '+':
3-          cout << a+b ;
4-          break;
5-    case '-':
```

```
6-          cout << a-b;
7-          break;
8-     case '*':
9-          cout << a*b;
10-              break;
11-         case'/':
12-              cout << a/b;
13-              break;
14-         default:
15-              cout << "Bad Command";
16-         }
```

أول شيء يجب النظر إليه أن تفرعات الجملة switch ليست مثل جمل if السابقة بل تبدأ بالكلمة المفتاحية case ، فمثلاً لو نظرنا إلى السطر الثاني فإن الأمر أشبه ما يكون هكذا:

```
        if ( x=='+')
```

الآن لنفرض أن المستخدم قام بإدخال العددين 5 و 6 وأدخل * كعملية حسابية ، وكما تعلم فإن المتغير x هو العملية الحسابية وهو المتغير الذي تقوم العبارة switch سيبدأ تنفيذ البرنامج وسينتقل إلى السطر 2 وكما ترى فإنه لا وجود للحالة الاولى بالنسبة للعملية * ، ينتقل التنفيذ بعد ذلك إلى الحالة الثانية في السطر 5 وكما ترى فليس هناك أي مطابقة وبالتالي فسينتقل التنفيذ إلى الحالة الثالثة في السطر 8 وكما ترى فإن هناك مطابقة بالفعل وبالتالي يدخل البرنامج في هذه الحالة التي يوجد لها أمران فقط الأول يطبع القيمة والأمر الثاني يطلب من البرنامج الخروج نهائياً وترك الجملة switch ومواصلة سير البرنامج بشكل طبيعي وهي الكلمة المفتاحية break ، في حال عدم وجود الكلمة break فإن البرنامج سيواصل التنفيذ وسيقوم بالدخول في الحالة الرابعة وبالطبع فلا وجود لمطابقة مع المتغير x وبالتالي ينتقل التنفيذ إلى الحالة العامة وسيقوم بتنفيذ أوامرها بالإضافة لتنفيذه أوامر عملية الضرب ، لذلك احرص دائماً على الخروج الآمن والسليم من العبارة switch .

محاذير حول الجملة switch :

ينبغي لنا هنا أن نتحدث قليلاً عن القيمة التي تقوم الـ switch بإختبارها ، تذكر أن ما تقوم هذه الجملة بإختباره هو المتغيرات وفقط ولا شيء آخر، لا تستطيع أن تقوم بكتابة أي تعبير لاختباره ، وقد ترى أن ذلك يقلل من قيمة switch إلا أن هذا غير صحيح فبإمكانك التوصل إلى نفس الهدف بطرق أخرى غير ما هو مفترض أو بديهي. اعتمد في هذا الأمر على تفكيرك أو حتى خيالك الواسع

ملاحظة مهمة:

تذكر أن الحالة default ليست حالة إستثنائية كما هو الأمر في الجملة else بل هي حالة عامة أي سيتم تنفيذها سواء كان المدخل صحيحاً (أي مطابقاً للحالات الأخرى) أو غير صحيح (أي غير مطابق للحالات الأخرى).

الكلمة break:

تستخدم الكلمة break للخروج الآمن والسليم من العبارة switch وتستخدم أيضاً في جمل التكرار وغيرها ، احرص دائماً حالما تنتهي من كتابة أي حالة من حالات switch أن تذيلها بالعبارة break فهذا سيجعل البرنامج يخرج من العبارة switch وبالتالي فلن يذهب للحالات الأخرى ولن يذهب حتى للحالة العامة للعبارة switch .

بقي أن نشير هنا إلى إحدى الملاحظات المهمة كما ترى فإن جميع حالات switch لم نقم بتغليفها في كتلة بين قوسين كبيرين { } بعكس الجملة else/if ، نصيحتي لك في النهاية أن تعمل بطرق البرمجة الآمنة وأن تبتعد عن مكامن البرمجة الخطرة ، ولا أعني حينما أقول البرمجة الخطرة بتلك الأخطاء التي تقع حينما تقوم بكتابة أوامر ثم يقوم المترجم بتصحيح الأخطاء لك ، بل أعني بتلك الأخطاء التي تظهر في التنفيذ لأسباب أخرى كثيرة ، وبعض الأخطاء لا يختص بأخطاء البرمجة من ناحية قواعدية syntax بل من ناحية خوارزمية أو من ناحية تصميمية أي تقوم بكتابة برنامج حتى يقوم بتنفيذ ما تريده أنت حقاً ويعطيك نتائج خاطئة وتعتبر هذه الأخطاء من أصعبها عندما تريد كشفها بالإضافة لأخطاء المؤشرات (سنتعرض لموضوع المؤشرات في وقت لاحق من الكتاب).

إستخدام المعاملات المنطقية مع الجملة if :

للمتغيرات المنطقية فوائد كثيرة للغاية ، إلا أنها تخفى عنا بسبب اعتمادنا الكبير على المتغيرات الأخرى وبسبب أيضاً أنها تأخرت قليلاً في الظهور في المترجمات المشهورة مثل البورلاند والفيجوال ، للمتغيرات المنطقية قيمتين فحسب واحدة منها هي صواب true والأخرى هي false ، ولن نقوم بوضع أي أمثلة عملية هنا بل سنترك الأمر كمهارة لك في المستقبل ، انظر لهذه الأسطر:

```
bool value=true;
 if (value)
cout << "Hellow";
```

قمنا بتهيئة المتغير المنطقي value بالقيمة true ثم تقوم الجملة if بإختبار الشرط وهو إذا كانت قيمة value صحيحة أو true ثم تقوم بطباعتها ، الجملة if شبيهة بالشرط التالي:

```
if (value==true)
```

أما في حال ما أردنا العكس فبإمكاننا كتابة التالي:

```
if (!value)
```

والتي تعني السطر التالي:

```
if (value==false)
```

لاحظ هنا أننا لم نقم بوضع علامتي أقواس الكتل الكبيرة { } والسبب في ذلك أننا لم نرد للجملة if أن تقوم سوى بتنفيذ جملة واحدة فحسب أما إذا أردنا كتابة أكثر من جملة فعلينا بتضمين الجمل أو الأوامر بين قوسين.

المعاملات المنطقية:

لم نناقش هذا الموضوع في الوحدة السابقة وليس السبب في ذلك عدم أهميته بل إن السبب يعود بالدرجة الأولى إلى تأجيل الموضوع لحين ظهور فائدته وبالتالي التأكيد على أهميته.

تستخدم المعاملات المنطقية كثيراً في الجمل الشرطية ، والسبب في ذلك إلى أنها تناور الجملة if وتجعلها تقبل أكثر من شرط مع العلم أن الجملة if لا تقوم بإختبار أكثر من شرط ولكن بواسطة المعاملات المنطقية فبإمكانك جعل أكثر من شرط شرطاً واحداً وبالتالي تستطيع مناورة الجملة if .

صحيح أننا قمنا بمناقشة بعضاً من هذا الموضوع في الوحدة السابقة إلا أننا لم نتعرض لثلاث معاملات أخرى مهمة وهي:

1- معامل (و) And : ورمزه && .
2- معامل (أو) OR : ورمزه || .
3- معامل (ليس) Not: ورمزه ! .

المعامل (And):

هذا المعامل يقوم باختبار تعبيرين وإذا كان كلاهما صحيحاً فإنه يرجع بالقيمة true ، لنفرض أنك تقوم بكتابة برنامج يقوم باختبار درجات الطلاب وإعطاؤهم التقدير المناسب ، فإنك ستكتب لحساب التقدير ممتاز هكذا:

```
if ( (total > 90) && (total < 100 ) )
```

وبالتالي فلن تعمل الجملة if إلا إذا كان التعبيرين صحيحين أما إذا كان أحدهما صحيح فلن تعمل .

المعامل (OR):

يقوم المعامل باختبار تعبيرين وفي حال كان أحد التعبيرين صحيحاً فإنه يرجع بالقيمة true ، لنفرض أنك تود إضافة جملة شرطية تقوم بالتأكد من أن المستخدم أدخل رقماً صحيحاً (نتحدث هنا عن برنامج درجات الطلاب) ، فإنك ستجعل الجملة if هكذا:

```
if ( (total < 0) || (total > 100 )
```

وبالتالي فستعمل الجملة if إذا أدخل المستخدم عدداً أصغر من الصفر وستعمل أيضاً إذا أدخل عدداً أكبر من 100 .

المعامل (NOT) :

يقوم هذا المعامل باختبار تعبير واحد وهي تعود بالقيمة true إذا كان التعبير الذي يجري اختباره خطأ ، لنفرض أنك تود كتابة برنامج يقوم المستخدم من خلاله بإدخال عددين اثنين ثم يتأكد البرنامج إن كان العدد الثاني ليس قاسماً للعدد الأول (ليكون قاسماً لا بد أن يكون خارج باقي القسمة يساوي الصفر) ، انظر لهذا الكود:

```
if ( !(numberOne% numberTwo == 0))
```

وبالتالي ففي حال كان خارج القسمة يساوي الصفر فلن يتم تنفيذ الجملة if .

مثال عملي:

سنقوم الآن بكتابة برنامج بسيط للطلاب يقوم الطالب فيه بإدخال درجته ثم يقوم الحاسب بإعطاءه التقدير (ممتاز أم جيد .. إلخ) .

وسنستخدم في هذا المثال العبارة if/else والمعاملات المنطقية وبالطبع ففي نهاية هذه الوحدة سنقوم بتطوير الكود ليقدم خدمات أكثر فائدة. وربما في المستقبل تستطيع تطويره ليصبح مشروعاً رسومياً متكاملاً.

CODE

```
1- #include <iostream>
2- using namespace std;
3-
4- int main()
5- {
6-        float degree=0;
7-        char mark;
8-
9-        cout << "Please Enter Your degree:\t" ;
10-             cin >> degree;
11-
12-             if ((degree <=100) && (degree>= 90))
13-                   mark='A';
14-
15-             else if ((degree <90) && (degree>= 80))
16-                   mark='B';
17-
18-             else if ((degree <80) && (degree>= 70))
19-                   mark='C';
20-
21-             else if ((degree <70) && (degree>= 60))
22-                   mark='D';
23-
24-             else if ((degree <60) || (degree>= 0))
25-                   mark='F';
26-
27-             else if((degree >100) || (degree < 0)) {
28-                   cout << "False degree" << endl;return 0;
29-             }
30-             else     {cout << "Bad command" << endl;
31-             return 0 ;}
32-             cout << endl;
33-             cout << "Your Mark:\t" << mark ;
34-             cout << endl;
35-
36-        return 0;
37-        }
```

في السطر 6 و7 قمنا بالإعلان عن متغيرين اثنين المتغير الأول هو درجة الطالب والمتغير الثاني هو تقدير الطالب.
في السطر 10 يطلب البرنامج من المستخدم إدخال درجته ثم ينتقل التنفيذ إلى عبارات else/if ، ولنفرض أن المستخدم أدخل كدرجة له العدد 102 وكما تعلم فإن هذه الدرجة غير صحيحة لأنها تجاوزت الدرجة النهائية وهي 100 ، وبالتالي فإن التنفيذ سيصل للجملة if التي تعالج هذا الوضع وهي موجودة في السطر 27 وهي كالتالي:

```
1- else if((degree >100) || (degree < 0)) {
2-              cout << "False degree" << endl;
3-              return 0;
4-       }
```

كما ترى فإن التعبيرين الذين تقوم الجملة else/if باختبارهما ، إذا ما كان أحدهما صحيحاً فستقوم بتنفيذ نفسها وإلا فستمنع البرنامج من تنفيذ السطر الثاني والثالث وكما ترى فإن التعبير الأول في حال ما أدخل المستخدم الدرجة 102 يعيد القيمة true وبالتالي يتجاهل البرنامج التعبير الثاني ولا يقوم باختباره أما إذا كان التعبير الأول يعيد القيمة false فلن يتجاهل التعبير الثاني وسيقوم باختباره ، بالنسبة لحالتنا الأولى فسيتم تنفيذ السطر الثاني والثالث ، وكما ترى ففي السطر الثاني يقوم البرنامج بطباعة الجملة False degree ثم حينما يصل للسطر الثالث يتم إنهاء البرنامج بواسطة الكلمة return 0 وهذا الفعل صحيح 100 % ولا يعيبه أي خطأ أو حتى تحذير من المترجم ، أما بالنسبة لإنهائنا البرنامج فيعود إلا أننا لا نريد من البرنامج أن يكون مضحكاً حاول أن تقوم بإلغاء السطر الثالث من الكود ثم أعد تنفيذ البرنامج وانظر مالذي سيحدث والنتائج الغريبة التي ستظهر.

بالنسبة لبقية عبارات else/if فلا جديد فيها وتقوم فقط باختبار الدرجة المعطاة وإظهار التقدير العام للدرجة.

الآن ما رأيك لو نخمن ما هي ادخالات المستخدم ، لنفرض أن المستخدم حاول أن يكتب في هذا البرنامج اسمه الكامل بدلاً من أن يقوم بإدخال درجته فهل تستطيع التخمين عما سيحدث في البرنامج.
لا أحد يعرف ما الذي سيحدث وقد تختلف النتائج من جهاز لجهاز ولكن حسب تخميني فقد ينتقل التنفيذ إلى الجملة else/if في السطر 24 . أما عن كيفية معالجة هذه الادخالات فبإمكاني تزويدك بإحدى التقنيات وإن كانت ناقصة فبإمكانك أن تكتب في أعلى جملة if ما يلي:

```
1-  if (cin.fail() ) {
2-              cout << "False degree" << endl;
3-              return 0;
4-       }
```

حيث يقوم التعبير cin.fail باختبار ما إذا كان الإدخال فشل. وأنا لا أعني حينما أقول بأن الإدخال فشل أي إدخال درجة الطالب بل أي إدخال آخر في البرنامج فأي متغير الآن تقوم بكتابته وإدخاله بواسطة تيار الإدخال cin فسيتم تطبيق الجملة if عليه وإنهاء البرنامج حتى وإن كان إدخال درجة

الطالب صحيح. ولا تعتقد أن الأمر ينتهي عند هذا الحد بل هناك أمور ينبغي علينا معالجتها من ضمنها آثار الخطأ الذي قام المستخدم بإدخاله لا تشغل بالك الآن بهذه الأمور فسيأتي وقتها فيما بعد.

الجملة goto :

لا أحد يستخدم هذه الجملة إلا إن كان فاشلاً أو هاوياً للبرمجة وأنا أعني الذي يستخدمها بكثرة وليس في حالات الضرورة القصوى جداً ، أما عن سبب وضعي فقرة لهذه الجملة فالسبب يعود إلى أنها ارتبطت تاريخياً بالتكرار مع العلم أنها ليست حلقة تكرارية بل هي جملة قفز تتنقل بين الأكواد ، أيضاً من أحد الأسباب أنها تعتبر مقدمة جيدة للغاية لموضوع حلقات التكرار ، سنقوم الآن بكتابة كود أيضاً للطلاب لكنه هذه المرة يطلب من المستخدم إدخال درجات خمس مواد ثم يقوم بحساب متوسط هذه المواد ، قد تعتقد أننا سنستخدم خمس متغيرات لكل مادة متغير ، لكن مع التكرار فلن نستخدم إلا ثلاثة متغيرات وسنختصر أكثر من 10 أسطر ، انظر إلى هذا الكود:

CODE

```
1- #include <iostream>
2- using namespace std;
3-
4- int main()
5- {
6-     float degree=0;
7-     float total=0;
8-     int i=0;
9-
10-    ss:
11-          cout << "Enter the degree of course number " << i+1
12-                << endl;
13-          cin >> degree;
14-          total=total+degree;
15-          i++;
16-          if (i<5)
17-          goto ss;
18-
19-          cout << "The Avreg is:\t"
20-                << total/5 << endl;
21-          return 0;
22-    }
```

انظر إلى السطور 6-8 تجد أنها ثلاث متغيرات ، المتغير الأول هو درجة المادة والمتغير الثاني هو مجموع المواد والمتغير الثالث هو الذي يقوم

بحساب عـدد المـواد التي قمت أنت بإدخالها وسيزيد هذا المتغير مرة واحـدة مع كل إدخال للمادة حتى يصل إلى الرقم أربعـة ثم يتوقف (يصل إلـى الـرقم 4 لأنـه يبدأ حساب عـدد الإدخالات من الرقم 0 وليس من الرقم 1).
دعــنا الآن نلقـي نظـرة فاحصــة علـى السـير الطبيعـي للبرنــامج ابتـداءً مـن السـطر العاشر:

- انظر إلى السـطر 10 تجـد أنــنا كتبنـا (ss:) يعتبـر هـذا الأمـر أشـبه مـا يكون بنقطـة قفز ستفهـم ما تعنيـه بعـد قليل .
- يسـتمر السـير الطبيعـي للبرنـامج حتـى يصـل إلـى السـطر 13 حيـث يطلب من المستخدم إدخال درجـة المادة الأولى .
- عـندما يصل البرنامج إلـى السـطر 15 فـإن المتغيـر i يزيـد مـرة واحـدة لأننا كما قلنا سابقاً أنـه مع كل إدخال يزيد العـدد i مرة واحدة .
- يدخل البرنامج في حلقـة if وسـينجح اختبـار الشـرط وبالتـالي ينتقـل التنفيذ إلى السطر 17 .
- يطلـب الكــود مـن البرنـامج الانتقـال إلـى مـا أسـماه ss عبـر الكلمــة المفتاحيـة goto يعـود البرنامج إلى السطر 10 ثم يعيد تكرار الأمر أكثر من أربع مرات.
- في المـرة الخامســة سـيكون المتغيـر i وصـل إلـى الـرقم 4 وبالتـالي فحينما يصل إلى السطر 15 سيزيد حتى يصل إلى الرقم 5 .
- لن ينجح إختبار الجملة if في المرة الخامسـة وبالتالي فلن يـتم تنفيـذ العبارة goto وسيستمر السير الطبيعي للبرنـامج.
- حينمـا يصـل البرنـامج إلـى السـطر 20 فإنـه يقـوم بقسـمـة مجمــوع المـواد على عـدد المـواد وبالتالي نحصل على المتوسـط الحسابي

الجمع التراكمي:
قد تسـتغرب مـن كتابـة السـطر 14 هــكذا وهـذا مـا يعـرف بـالجمع التراكمـي فلنفـرض أن البرنـامج لـم يـزال فـي المـرة الأولـى كمـا تعلـم فـإن قيمتـي المتغيرين grade و total صفر ، حينما يقوم المستخدم بإدخال قيمـة الـمتغير grade وينتقل التحكم إلى السطر 14 كما يلي:

```
total=total+degree;
```

فإن البرنامج يقوم بجمع قيمـة المتغير grade مع قيمــة المتغيـر total والتـي هي حالياً صـفر ثـم يأخـذ مجمــوع المتغيـرين ويضيفهــما إلـى نفـس المتغيـر total وهذا ما يعرف بالجمع التراكمي ، إذا ما أدخل المسـتخدم الدرجــة 100 إلى المتغير grade فإن قيمـة المتغير total تصبح 100 ، في المرة الثانيـة إذا قام المستخدم بإدخـال القيمــة 20 إلـى المتغيـر grade فـإن البرنـامج حينمـا يصل إلى السطر 14 فإنـه يأخذ قيمـة المتغير grade التي أدخلها المسـتخدم ويضــيفها مــع المتغيــر total والــذي هــو حاليــاً 100 (حســب دورة التكــرار السابقة) إلى نفس المتغير total لتتغير من 100 إلى 120. ونفس ما يحـدث سيحدث في الـدورات التكراريـة القادمــة، وهــكذا فـإن المتغيـر grade يتغيـر دائماً وسيستخدم المتغير total لمراكمـة إدخالات المتغير grade من هنا أتى مسمى الجمع التراكمي ، أما إذا ما أردت القيام بتخـزين درجات جميع المـواد فالتقنيــة الوحيـدة هــي المصـفوفات أو القـوائم المترابطــة أو مـا ستتعلمــه لاحقاً.

محاذير بالنسبة للجملة goto :

كما ترى فإن البرنامج يقوم بتجاوز النقطة ss عند بداية تنفيذه ، تخيل لو كان لديك أكثر من نقطة وأكثر من جملة goto وستتداخل جمل goto في بعضها حتى يصبح من المستحيل متابعة البرنامج وقد تظهر أخطاء منطقية قد يكون من العسير كشفها إن لم يكن شبه مستحيل ، لذلك قام المبرمجين بتشبيه البرامج التي تحتوي على الكثير من جمل goto ببرامج معكرونة الأسباجيتي . سنتعرف الآن على أول حلقة تكرارية وهي do/while .

الجملة do/while :

قد يتساءل البعض عن السبب وراء البدء في موضوع الحلقات التكرارية بالجملة do/while بدلاً من الحلقات الأخرى ، والسبب في ذلك يعود إلى أن هذه الحلقة قريبة للغاية من الكود السابق وبالتالي الجملة goto مما سيسهل الكثير من الشرح والفهم.

الصيغة العامة لهذه الحلقة هي كالتالي:

```
do
  {
     statement1;
     statement2;
       }  while (expression) ;
```

بإمكاننا القول أن الحلقة do/while تعني قم بالدخول في الكتلة do وقم بتنفيذ الأوامر وفي حال الانتهاء قم باختبار التعبير الذي لدى الكلمة while وفي حال صحته قم بالرجوع إلى مكان الكلمة do .

بإمكاننا تعديل الكود السابق والاستغناء عن جميع جملة if وجملة goto ، انظر لهذا التعديل في الكود السابق:

```
1- do {
2-     cout << "Enter the degree of course number " << i+1
3-           << endl;
4-     cin >> degree;
5-     total=total+degree;
6-     i++;
7-     } while (i<5);
```

وبالرغم من أن استخدام هذه الحلقة قليل إلا أن ذلك لا يقلل من قيمتها وفائدتها العظيمة. وسترى أنه في بعض الأحيان لا يمكنك حل معضلة برمجية إلا بواسطة هذه الجملة.

مثال عملي:

سنقوم الآن بكتابة برنامج يقوم بكتابة جدول الضرب لأي رقم تود إظهاره ، وبالطبع سنستخدم فيه الحلقة do/while .

CODE

```
1- #include <iostream>
2- using namespace std;
3-
```

```
4- int main()
5- {
6-     double number=0;
7-     int i=0;
8-     cout << "please Enter The Number:\t";
9-     cin >> number;
10-          cout << endl << endl;
11-          cout << "Number\t\tOther\t\tValue"<< endl;
12-          do
13-          {
14-                cout << number << "  \t\t";
15-                cout << i      << "  \t\t";
16-                cout << i*number;
17-                cout << endl;
18-                i++;
19-          } while ( i<=10);
20-          return 0;
21-   }
```

في السطر 9 يقوم البرنامج بالطلب من المستخدم إدخال الرقم الذي يريد طباعة جدول الضرب لديه بالنسبة للسطرين 10 و 11 فهي تقوم بتحسين المظهر العام للجدول.
يدخل البرنامج في الحلقة do/while وتقوم الأسطر 14-17 بطباعة العددين المضروبين والناتج وتحسين المخرجات كذلك أما السطر 18 فهو يقوم بزيادة العدد الآخر المضروب زيادة واحدة حتى يستطيع البرنامج ضرب العدد الذي قمت بإدخاله في عدد آخر وتختبر الجملة while فيما إذا كان المضروب الآخر أقل من 11 وإلا فإنها ستخرج من الحلقة وبالتالي تخرج من البرنامج.
لقد انتهينا من الحلقة do/while وقد تركنا بعض المواضيع للحلقتين التاليتين وهما for و while .

الحلقة while :

هناك فرق بين الحلقة while والحلقة do/while ففي الأخيرة يدخل البرنامج في الحلقة ثم يصطدم بالشرط أو التعبير وينتظر اختبار الشرط ، فإن كان صحيحاً أعاد التكرار مرة أخرى وإن خاطئاً استمر البرنامج في عمله دون توقف ، أما في الحلقة while فإن البرنامج يصطدم بالشرط أولاً قبل أن يدخل الحلقة ، أنظر الصيغة العامة لهذه الحلقة:

```
while (expression) {
            statement1;
            statement2;
            statement3;
               }
```

سنقوم الآن بكتابة مثال كـودي بسيط حيث نطلب من المسـتخدم فيـه كتابـة ما يريد وفي حال وجد البرنامج علامـة النقطـة فإنـه ينتهي.

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.     char d='a';
7.     cout << "Please Enter What You want \n";
8.
9.     while (d!='.'){
10.                cin >> d;
11.          }
12.
13.          cout << endl << "Finish" << endl;
14.
15.          return 0;
16.    }
```

انظر إلـى السـطر 9 ، تجـد أن الشـرط هـو عـدم إسـناد المحـرف (.) إلـى المتغير الحرفي d وفي حال وقع ذلك فإن البرنامج يخرج من التكرار while .

بإمكانك تطوير المثال الحالي حتى يصبح قادراً على عـد الحروف المدخلة.
وبإمكانك أيضاً تحـويل أمثلة التكرار do/while إلى الحلقة while .

ليس في المثال الحالي أي زيادة عـددية ، سنقوم الآن بكتابة مثال آخر يقـوم بعرض الاعداد من 0 إلى 10:

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.     int number=0;
7.
8.        while (number <=10) {
9.           cout << "The number is :\t";
10.                cout << number;
```

```
11.                 cout << endl;
12.                 number++;
13.           }
14.           return 0;
15.     }
```

حاول أن تفهـم المثال أعلاه بنفسك من دون أي شرح ، ثم انتقل إلـى المثـال القادم .

مثال عـملي:

سنقوم الآن بكتابة كـود يقـوم بعـرض الأعـداد الزوجيـة مـن أي عــدد يقـوم المستخدم بتحـديده إلى أي عـدد يقوم المستخدم بتحـديده أيضاً.

هـناك مسائل يجب أن نتناولها بعـين الحذر فماذا لو قرر المسـتخدم أن يـدخل عـدداً فردياً ، لذلك علينا أن نتأكد من أن أول عـدد هـو عـدد زوجي وفـي حـال لم يكن فعلينا بزيادته عدداً واحداً حتى يصبح زوجياً ، انظر لهذا المثال:

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.    int number=0;
7.    int max=0;
8.    int min=0;
9.
10.
11.          cout << "Please Enter The First Number:\t";
12.          cin >> min;
13.
14.          cout << "Please Enter The Last Number:\t";
15.          cin >> max;
16.
17.          if (!(min%2==0)) min++;
18.
19.         number=min;
20.
21.                 while(number < max) {
22.                 cout << "The Next Number is\t";
23.                 cout << number << endl;
```

```
24.             number=number+2;
25.         }
26.
27.         return 0;
28.   }
```

هـنـاك ثلاثــة أعـداد فـي الأسـطر 6 و 7 و8 ، أحدهــما هــو أول عــدد يقـوم المستخدم بإدخاله حتى يبدأ البرنامج منــه لعــد جميـع الأعـداد الزوجيــة أمـا العـدد الآخر فهـو عـدد يقوم المستخدم بإدخاله حتى ينتهي العـد عـنده ، أمـا المتغيـر الثالـث فهــو العـــدد الـذي يسـتعمله البرنـامج للتنقـل بـين الأعـداد الزوجية؛ وبالطبع فإن مكمن الخــطورة هــنا هــو أول عــدد يقـوم المســتخدم بإدخاله فهذا العـدد لو كان فردياً وابتدأ العد منـه لأصبحت جميع الأعداد التي سيخرجها البرنامج أعداداً فردية.
فكرة هذا المثال تقـوم علـى التاكـد مـن أن أول عــدد هــو زوجـي ثـم إضافة الرقم 2 إليه وطباعـة العـدد الجديد وهـكذا حتى يصل هذا العــدد إلــى العــدد الأخير.
يقوم السـطر 17 بالتأكد أن العـدد المدخل الأول هـو عدد زوجي وفي حال لـم يكن كذلك فإنه يضيف إليه الرقم واحد حتى يصبح زوجياً.
يقوم السـطر 19 بإسناد قيمـة العـدد الاول إلى العداد الـذي سـيبدأ البرنـامج العـد منـه وهـو المتغير number .
تبدأ الحلقـة while من السطر 21 إلى السـطر 25 ، تـتم الزيـادة فـي السـطر 24 حيث يزيد العـداد مرتين وليس مرة واحدة.
تنتهي الحلقـة while حينما يختل شـرطها وهــو أن يكـون العـداد أكبـر مـن أو يساوي العـدد الاكبر.

الحلقة : for

الحلقـة for من الممكن تشبيهها بأنها عـداد ينتهـي عـنـد وصـول هـذا العــداد إلى رقم معـين ثـم ينتهـي بعــكس الحلقـة while والتـي هــي تقـوم بتكـريـر نفسها ما دام الشـرط محققاً ، تأخذ الحلقـة for الصيغـة التالية:

```
for ( expr1 ; expr2 ; expr3) {
          statement1;
          statement2;
          statement3;
         }
```

حيث أن:
expr1 : هـو القيمـة الابتدائيـة للتكرار.
expr2 : وهـو الشرط.
expr3 : وهـو الزيادة بعـد كل دورة.

سنقوم الآن بكتابة مثال يقوم بعـد الأعداد من 0 إلى 10 حتـى يفهــم القـارئ ما تعـنيه الصيغـة العامـة للحلقة for ، وهذا الكـود هـو إعـادة صياغــة المثـال السـابق.

```
1. #include <iostream>
2. using namespace std;
3.
```

```
4. int main()
5. {
6.    int number;
7.
8.    for (number=0;number <=10;number++) {
9.          cout << "The number is :\t";
10.               cout << number;
11.               cout << endl;
12.         }
13.         return 0;
14.   }
```

مثال عـملي:

سنقوم الآن بكتابة مثال يقوم بجعل المستخدم يقوم بكتابة عشـرة أرقـام ثـم يقوم البرنامج باختيار أكبر رقم وأصغر رقـم ووسيلتنا لفعل ذلك هــي الحلقـة for بالإضافة للجملة if .

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.    int number=0;
7.    int max=0;
8.    int min=0;
9.
10.
11.         for (int i=0; i< 10;i++) {
12.               cout << "Enter the number:\t";
13.               cin >> number;
14.
15.               if (number > max)
16.                     max=number;
17.
18.               if (number < min)
19.                     min=number;
20.         }
21.
```

```
22.          cout << endl << endl;
23.          cout << "The Max Number is:\t" << max;
24.          cout << "\nTne Min Number id:\t" << min;
25.          cout << endl;
26.
27.          return 0;
28.
29.   }
```

- هناك ثلاثة متغيرات هي العدد الأكبر max والعدد الأصغر min والعدد الذي سيقوم المستخدم بادخاله وهو number وأيضاً هناك العداد وهو المتغير i .
- تبدأ الحلقة for في السطر 11 وستستمر في الدوران 10 مرات ، حسب شرط الحلقة for .
- الآن سيطلب البرنامج من المستخدم إدخال العدد الأول ، ثم يقوم بالمقارنة إن كان أكبر من العدد الأكبر max وفي حال كان ذلك فإنه يسند قيمته إلى المتغير max ، وهذا كله في السطرين 15 و 16 .
- ثم يقارنه أيضاً بالمتغير min وفي حال كان أصغر فإنه يسند قيمته إلى المتغير min .
- في الدورة الثانية يقوم المستخدم بإعادة إدخال العدد number وتستمر المقارنة حتى يخرج من البرنامج وبالتالي تتم طباعة العدد الأكبر والأصغر في السطرين 23 و 24 .
- قد تستغرب من السطر 11 ، حيث قمنا بالإعلان عن المتغير i ضمن الحلقة for ، وذلك صحيح قواعدياً في لغة السي بلس بلس ، وبإمكانك الإعلان عن المتغيرات في أي مكان في لغة السي بلس بلس بعكس لغة السي والتي تلزمك بأن تصرح عن المتغيرات في رؤوس التوابع.

لزيادة فهمك في الحلقات التكرارية قم بإختراع أمثلة جديدة.

سنتعرف في وحدة التوابع على بديل جديد ولكنه أقل أهمية وفائدة وهو العودية.

الجملة break :

تستخدم الجملة break في الخروج من الحلقات التكرارية ، وأيضاً من جملة switch ، ولكنها لا تستخدم مع الجملة if وتفرعاتها.
تقوم الجملة break بإنهاء الحلقة التكرارية قبل إكمال الشرط وهذا له فائدة كبيرة جداً ، وأيضاً هي تفيدك في الخروج من الحلقات التكرارية الأبدية.
تأتي هذه الجملة في الخطورة بعد الجملة goto بالإضافة إلى الجملة continue والسبب في ذلك يعود إلى أنها تقفز وتخرج مما يؤدي في بعض الأحيان إلى صعوبة تتبع سير البرامج.
حتى تفهم الفائدة من الجملة break فسنقوم الآن بكتابة كود يقوم باختبار العدد الذي تقوم باختياره ويرى إن كان عدداً أولياً أم لأ.
فكرة هذا المثال الكودي تقوم على أن البرنامج سيقوم بقسمة الأعداد من العدد الذي قبله وحتى رقم 2 وفي حال كان خارج باقي قسمة هذين

العـدين يساوي الواحد فإن البرنـامج سـيخرج ويخبـر المسـتخدم بـأن العـدد غير أولي ،انظر إلى هذا الكـود وحاول أن تفهـمـه قبل أن تقرأ شرحـه.

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.     int number=0;
7.
8.     cout << "Please Enter The Number:\t";
9.     cin >> number;
10.
11.         for (int i=number-1 ; i>1 ; i=i-1)
12.         {
13.               if (number%i==0)
14.                     break;
15.         }
16.
17.         if (i==1)
18.               cout << endl << "The Number  are " ;
19.         else  cout << endl << "The Number not are";
20.
21.         cout << endl;
22.
23.         return 0;
24.   }
```

هـنـاك متغيـران فـي البرنـامج فحسـب ، الأول هـو العـدد الـذي سـيختبره البرنامج إن كان أولياً أم لا ، والثاني هـو عـداد الحلقـة for ، يـدخل البرنـامج في الحلقـة for فـي السـطر 11 ، يبـدأ العـدد مـن العـدد الـذي قبـل العـدد المختبر وتقوم هذه الحلقة بقسمة العدد المختبر الـذي أدخلـه المسـتخدم على عـداد الحلقة وتستمر القسمة حتى يصل العداد إلى القيمة 1 ، وفي حال وصوله فإن البرنامج سيخرج من الحلقة ولن يقسم العدد المختبر علـى العدد 1 ، في حال ما إذا كان خارج القسمة مع أي رقم من العدد الذي قبـل العـدد المختبر إلى العـدد 2 فـإن البرنـامج سـيخرج مـن الحلقـة دون إكمالهـا وسـينتقل التنفيـذ إلـى السـطر 17 ، وسـتختبر الجملـة if العـداد فـإذا كـان مساوياً الواحد فإن ذلك يعـني أن العـداد أو الحلقة استمرت فـي القسـمـة حتى وصلت للعدد 1 ، ولم تجد أي عـدد خارج قسمته يساوي صفر وبالتـالي فإن العـدد أولي ، وستطبع رسالة بهذا الشأن أما إذا خرجـت الحلقـة قبـل أن

يصل العداد إلى الرقم 1 ، فسينتقل التنفيذ إلى السطر 19 وسيطبع البرنامج رسالة بأن هذا العدد ليس أولياً.

الجملة continue :

تتسخدم الجملة continue ليس للخروج من البرنامج كما هو الحال في الجملة السابقة بل لأجل إعادة التكرار ، فإذا ما وجد البرنامج الكلمة continue في حلقة التكرار فإنه يقوم بالتكرار مباشرة دون النظر إلى الأسطر المتبقية في البرنامج.

سنقوم الآن بكتابة كود يطبع الأعداد الزوجية فقط إلى أي عدد يحدده المستخدم ابتداءً من الصفر ، وفكرة هذا الكود بسيطة وليست مثل الكود السابق:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.    int number=0;
7.
8.    cout << "please Enter the number:\t";
9.    cin >> number;
10.
11.          cout << endl << endl;
12.
13.          for (int i=0 ; i<=number ; i++)
14.          {
15.                if (!(i%2==0))
16.                      continue;
17.                cout << "The number is: " << i ;
18.                cout << endl;
19.          }
20.
21.
22.
23.          return 0;
24.   }
```

يقوم المستخدم في السطر 9 بإدخال العدد الذي يريد البرنامج التوقف عن طباعة الأعداد الزوجية عنده.

يدخل البرنامج في الحلقة for وسيقوم بالعد من الصفر حتى العدد الذي أدخله المستخدم.

في السطر 15 يقوم البرنامج بإختبار ما إذا كان العدد الذي وصلت إليه الحلقة for فردياً وفي حال كان فردياً فإن التنفيذ سينتقل إلى السطر 16 أي

إلى الجملة continue والتي ستتجاهل بقية الأوامر في الحلقة for (أي السطر 17 و18) وتستمر في جعل الحلقة for تستمر
أما في حال لم يكن العدد المدخل فردياً فسيستمر تنفيذ الأسطر 17 و 18 دون أية مشاكل.

المعامل الشرطي الثلاثي ؟ :

يعتبر هذا المعامل هو المعامل الوحيد الثلاثي الموجود في لغة السي بلس بلس وهو شبيه للغاية بالجملة if/else ، أفضل وسيلة لشرح هذا المعامل هو تمثيله في كود بسيط
سنقوم بكتابة كود يقوم بإيجاد القيمة المطلقة لأي عدد تدخله.

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4.
5. int main()
6. {
7.    int Number=0;
8.
9.          cin >> Number;
10.         cout << "The Abs Value" << endl;
11.         int Abs = Number < 0 ? -Number : Number;
12.         cout << Abs << endl;
13.
14.         return 0;
15.   }
```

هناك متغيران الأول هو المتغير Number والذي سيدخله المستخدم والمتغير Abs الذي سيتم إيجاد قيمة Number المطلقة وتخزينها فيه.
انظر إلى السطر 11 إن معنى هذا السطر:

```
      int Abs=Number < 0 ? -Number : Number;
```

أي قارن المتغير Number بالعدد صفر فإذا كان أصغر فقم بجعل المتغير سالباً وقم بإسناد القيمة الجديد إلى المتغير Abs وإلا فقم بإسناد نفس قيمة المتغير Number دون أي تغيير إلى المتغير Abs .
الآن سنقوم بكتابة نفس الكود السابق ولكن هذه المرة باستخدام الجملة if/else :

CODE

```
1. #include <iostream>
2. using namespace std;
3.
```

```
4.
5. int main()
6. {
7.    int Number=0;
8.          cin >> Number;
9.    cout << "The Abs Value" << endl;
10.         int Abs;
11.         if (Number < 0) Abs=(-Number);
12.         else Abs=Number;
13.         cout << Abs << endl;
14.
15.         return 0;
16.   }
```

تعرف على المكتبة cmath :

سنتعرف الآن على إحدى المكتبات التي أتت ورثتها لغة السي بلس بلس من السي وهي cmath .

التابعان floor و ceil :

يستخدم هذان التابعان للأعداد العشرية حيث يقومان بتقريبهما إلى عدد صحيح، يقوم التابع floor بتقريب العدد العشري إلى أقرب عدد صحيح قبل العدد العشري فمثلاً لو كان لدينا العدد 12.9 وقمت بتمرير العدد كوسيط له فسيقوم بإعادة العدد 12 ، الكلمة floor مأخوذة من معنى السقف ، أما التابع ceil فهو عكس التابع السابق فهو يقوم بتقريب العدد العشري إلى عدد أكبر منه فلو استخدمنا العدد 12.1 لأعاد التابع ceil أكبر عدد صحيح بعد العدد العشري السابق ألا وهو 13 . انظر إلى هذا المثال الكودي لاستخدام هذان التابعان المفيدان.

CODE

```
1. #include <iostream>
2. #include <cmath>
3. using namespace std;
4.
5. int main()
6. {
7.    double Num=0 ,Val=0 ;
8.
9.    cout << "Enter The Num:  ";
10.         cin >> Num;
11.
12.         Val=floor(Num);
```

```
13.         cout << "Number (floor)  " << Val << endl;
14.
15.         Val=ceil(Num);
16.         cout << "Number (ceil)  " << Val << endl;
17.
18.         return 0;
19.   }
```

حاول أن تفهـم الكـود السابق وفائدة التابعان floor و ceil .

التابعان pow و sqrt :

يستخدم التابع pow كأداة قوية إذا ما أردت حسـاب الأس أو القوة لعـدد ما، يتم إستخدام هذا التابع هـكذا:

```
Number= pow (Number , power) ;
```

حيث العـدد Number هـو العـدد الذي تريد رفعـه والعـدد power هـو القـوة أو الأس الذي تـود رفع العـدد Number إليها.

أما التـابع sqrt فيحسـب لـك الجـذر التربيعـي للعـدد ، ويسـتخدم هـذا التـابع هـكذا:

```
Number= sqrt (Number);
```

حيث العـدد Number هـو العـدد الذي تـود حساب جذره التربيعي ، انظر إلـى هذا المثال الكـودي:

CODE

```
1. #include <iostream>
2. #include <cmath>
3. using namespace std;
4.
5. int main()
6. {
7.    double Num=0 ,Val=0 ;
8.
9.    cout << "Enter The Num:  ";
10.         cin >> Num;
11.
12.         Val=sqrt(Num);
13.         cout << "Number (sqrt)  " << Val << endl;
14.
15.         cout << "Enter The Power  " ;
16.         cin >> Val;
17.
18.         Val=pow(Num , Val);
```

```
19.         cout << "Number (pow)  " << Val << endl;
20.
21.         return 0;
22.   }
```

توليد الأعداد العشوائيـة rand( ) :

بإمكانك توليد الأعداد العشوائيـة بواسطـة التابع rand **الموجود فـي المكتبـة** iostream **، فإذا مـا أردت إسـناد عـدد عشـوائي إلـى متغيـر فاسـتخدم هـذه الصيغـة.**

```
      int Number= rand( ) ;
```

أيضاً هـناك طريقـة أفضل.

إذا أردت مثلاً إسـناد عــدد مـن 0 **إلـى** 6 **يختـاره الحاسـب عشـوائياً فبإمكانـك استخدام معامل باقي القسمـة** % **، انظر:**

```
int Number= (rand( )%5)+1;
```

المصفوفات والسلاسل

Arrays And Strings

بداية:

لنفرض أنه طلب منك كتابة برنامج بسيط للغاية وهو إدخال درجات عشر طلاب ؛ لكي تحل هذا البرنامج فإن عليك أن تقوم بالإعلان عن 12 متغيراً من نوع float ؛ وربما أن هذا مقبول نوعاً ما ؛ ولكن ماذا لو طلب منك إدخال أكثر من درجات 1000 طالب لحل هذه الإشكالية توفر لك لغة السي بلس بلس المصفوفات.صحيح أننا قمنا بحل مسائل من هذا النوع لم تتطلب المصفوفات لكن ماذا لو طلب منك البحث عن درجة طالب معين فلن يكون هناك أي حل إلا بواسطة المصفوفات.

تعريف المصفوفات:

هي عبارة عن مجموعة من البيانات التي تشترك في الاسم والنوع ولكنها تختلف في القيم المسندة إليها

الإعلان عن المصفوفة:

انظر إلى هذا السطر

```
int mark[10];
```

السطر السابق هي طريقة الإعلان عن المصفوفة وكما تلاحظ فإن الإعلان يحوي ثلاثة أشياء:

نوع المصفوفة ؛ واسم المصفوفة ؛ وعدد عناصر المصفوفة.

عدد عناصر المصفوفة يجب أن يكون بين قوسين [ ].

نوع المصفوفة	اسم المصفوفة	عدد عناصر المصفوفة	
int	**mark**	**[20]**	**;**

أعضاء المصفوفة:

في المصفوفة السابقة ؛ فإنها تحوي هذه العناصر:

int mark[0] ;	**int mark[1] ;**	**int mark[2] ;**
int mark[3] ;	**int mark[4] ;**	**int mark[5] ;**
int mark[6] ;	**int mark[7] ;**	**int mark[8] ;**
int mark[9] ;		

كما تلاحظ فإن المصفوفة مكونة من عشرة عناصر حسبما هو مكتوب في الإعلان السابق . ألا ترى الرقم الملون بالأزرق هذا الرقم هو ما يسمى بدليل المصفوفة والذي يميز بين عناصر المصفوفة الواحدة ؛ المميز هنا هو أن أول عنصر في المصفوفة هو int mark[0] وآخر عنصر هو int mark[9] وكما تلاحظ فإنه لا وجود للعنصر العاشر وهذا ما عليك أن تعرفه وهو بالغ الأهمية العد في المصفوفة يبدأ من العنصر رقم صفر وينتهي إلى العدد ما قبل الأخير من عدد أعضاء المصفوفة المعلن عنه.

الـوصول إلى عـناصر المصفـوفـة:

حسـب الشـكل التوضـيحي السـابق فإنـك تسـتطيع الوصـول إلـى أي عــنـصر فـي المصفـوفـة عبر كتـابة نـوع المصفـوفـة واسمها ثم دليل العـنصر فمثلاً للوصول إلـى أول عـنـصر في المصفوفة تستطيع كتابة int mark[0] وكمـا تلاحـظ مجـدداً فـإن أول عـنـصر فـي المصفوفـة دليلـه هـو صفر ؛ دعـنا الآن مـن هـذا الكـلام النظـري ودعـنا ندخل لمرحلة الكتابة الكوديـة:

مثال عـملي:

سنقوم بكتابة كـود للطلاب،عدد الطلاب فيه هـو عشرة ، ثم نحسب متوسط درجات هــؤلاء الطـلاب ، لـذلك نسـتطيع الإعـلان عــن مصفوفـة مكـونة مـن عشر عــناصر وسنقوم بتسـميتها int stud[10] ؛ بعـد ذلك نطلب من المستخدم إدخـال درجـات الطلاب بواسطـة دالة تكراريـة ونسطيع الإعلان عــم متغير مـن نـــوع int وآخر مـن نـوع float حيـث أن مهـمــة الأول هــي حسـاب مجمــوع درجـات الطـلاب والثـاني وظيفتـه قسـمـة المجمــوع علـى عـدد الطـلاب ؛ وهـكذا انتهينا مـن حل المشـكلـة وبقي أن نحـول الحل إلى كـود وهـو كالتالي:

CODE	
1	#include <iostream.h>
2	main ( )
3	{
4	int stud[10] ,total=0 , i ;
5	float Avrege;
6	cout << "Please Enter all grades of stud:\n" ;
7	for (i=0 ; i<10 ; i++)
8	{
9	cout << "grade number" << i+1 << endl;
10	cin >> stud[i] ;
11	total=total+stud[i] ;
12	}
13	Avrege=total /10;
14	cout << "The Avrege of all student is: " << Avrege ;
15	return 0;
16	}

بهذه الطريقـة يمكن حل السؤال السابق كما تلاحـظ فلقد إسـتخدمنا متغيـر من نـوع int هـو i والسبب في ذلك كما تـرى هــو دالـة for ؛ فكمـا تلاحـظ أن دليل المصفـوفـة في الدالة التكرارية هـو i ؛ والذي يزيد بعد كـل إدخـال مـرة واحدة وبالتالي ينتقل البرنامج من العـنصر الاول إلى العــنصر الثـاني وحتـى آخر عــنصر وكما تلاحظ أيضاً إسـتخــدمنا متحــول total والـذي يقـوم بحسـاب مجمـوع الدرجات فهـو أولاً يسند أول عــنصر من المصفـوفــة إلـى نفســه ثـم فـي الــدورة التكراريــة الثانيــة يقـوم بإسـناد مجمــوع العــنصر التـالي مـن

المصفـوفة ومجمـوعه هـو أيضاً إلى نفسـه ويستمر هكذا حتى الخـروج مـن دالة for .

> **تحذير:**
> لا تحـاول أبداً في البرنامج السابق أن تغيـر الشـرط فـي الدالـة التكرارية for من i<10 إلى مثلاً i<12 فذلك لن يزيد مـن حجـم المصفـوفـة ولـن يفعـل أي شيء لك ؛ فقط كل الذي سيفعله البرنامج أنه سيكتب العنصر الحادي عشر فـي مكـان خـارج حـدود المصفـوفـة أي فـي ذاكـرة أخــرى غيـر مخصصـة للبرنامج ربما تكون هذه الذاكرة مخصصة لبرنـامج آخـر أو لنظـام التشـغيل أو لأي شـيء مهـما كان ؛ وقد لا يكون كذلك فربما أن ذلـك سـيؤثر علـى برنـامج ولن يعـمل ..... أيضاً أحد الأخطاء الشائعـة هـو كتابة الشرط هكـذا i>10 هـذا الشـرط سيؤدي إلى عـدم توقف برنامجك نهائياً لذلك لا تحاول أن تجربـه

تهيئة المصفوفات:

بإمكانك إدخال عـنصر المصفوفـة دون الحاجـة إلى دالة for وذلك عبر تهيئتها من داخل برنامج فمثلاً بإمكـانك كتابة السـطر التالي:

```
int mark[7] = { 5,10,90,100,90,85,15};
```

وهذه الطريقـة في حال أنك لا تريد أن يدخل المستخدم أي أرقام للمصفوفـة.
وعبر هذه الطريق بإمكـانك الإستغـنـاء عـن عـدد عـناصر المصفوفـة الموجـود بـين قوسين ؛ هكذا:

```
int mark[] = { 5,10,90,100,90,85,15};
```

وسيقوم المترجم بعد العـناصر الموجودة في المصفـوفة.
لكن ليس بامكانك كتابة السطر السابق لكي تطلب مـن المسـتخدم إدخـال عــناصر المصفوفـة.
لاحظ: أنه في جميع طرق الإعلان عــن المصفـوفـة فلا بـد عليـك مـن تحديـد حجـم المصفوفـة وإلا فإن المترجم سيعـطيك خطأ.

أنـواع المصفوفات:

كما هـو معلوم فإن المصفوفـات نـوعـان:

1. المصفـوفـة الأحاديـة: وهـي مكونـة من بعد واحد فقط.
2. المصفوفـة المتعددة الأبعاد: وهـي مكونـة من عـدة صفوف وأعـمدة (لـيس شرطاً أن تكون بعدين)

المثال السابق عبارة عـن مصفوفـة من بعد واحد.
طريقة الإعلان عـن المصفوفة متعددة الأبعـاد هـي نفسـها فـي طريقـة المصفوفـة الأحاديـة غير أنك هذه المرة ستضع بعـداً آخر كالتالي:

```
int mark[10] [8];
```

وكمـا تلاحـظ فـإن الإعـلان السـابق هــو لمصفـوفة ذات بعـدين مكونــة مـن عشـرة صفوف وثمانية أعـمدة.

مثال كـودي:

سنقوم بكتابة برنامج لإدخال رواتـب خمـس مـوظفين فـي ثلاثــة أقسـام ثـم نقوم بحساب متوسط كل قسم ثم نحسب متوسـط رواتـب جميـع المـوظفين في جميع الأقسام

من الملاحظ أن هذا البرنامج لن تستطيع حله إلا باستخدام مصفوفة ثنائية البعد حيث عدده صفوفها ثلاثة وهو عدد الأقسام وعدد الأعمدة خمسة وهو عدد الموظفين ؛ وللإستمرار في حل البرنامج فسنقوم بإنشاء مصفوفة أحادية جديدة تحوي مجموع رواتب كل قسم ونحن لا ننشيء هذه المصفوفة لأن حل مشكلة البرنامج هي هكذا بل لتسهيل الحل والفهم ؛ تذكر المصفوفة الجديدة ستكون مكونة من ثلاثة عناصر.

CODE

```
1        #include <iostream.h>
2        main( )
3        {
4        int employee[3][5] , size[4] , i , j , sum=0 ;
5        size [3]=0;
6       cout << "Please Enter all employees salary" << endl;
7
8       for (j=0 ; j < 3 ; j ++ )
9       {     cout << " Enter the department " << j+1 << endl;
10           for (i=0 ; i < 5 ; i++ )
11                       {
12                          cout << " Employee number " << i+1 << endl;
13                         cin >> employee[ i ] [ j ] ;
14                         sum= employee[ i ] [ j ] + sum ;
15                       }
16             size[i] = sum/5 ;
17                cout << " The avreg is" << size[i] << endl;
18              cout <<"________________________________";
18             size[3] = size[3] + size [i];
19             sum=0
20             }
21        cout  <<  "  The  avrege  of  all  salary  of  employee  is:  "  <<
size[3] << endl;
22     return 0;
23     }
```

دعنا نبدأ بأول ملاحظة وهي وجود دالتي for وليس دالة واحدة كما في المثال السابق ؛ السبب في ذلك أن دالة for الأولى تقوم بتثبيت رقم الصف فيما تقوم دالة for الأخرى بتثبيت رقم العمود والذي يتغير بإستمرار حتى يتوقف تحقيق شرط الدالة الثانية وهو i < 5 والذي يفعله البرنامج هو أنه سيقوم بالخروج من دالة for الثانية ويكمل سير البرنامج وهو الآن ما زال

في الدالة for الأولى ويقوم بتنفيذ الأسطر من 16 إلى 19 ؛ كما تلاحظ في السطر السادس عشر يسند البرنامج قيمة متوسط حساب الموظفين إلى أول عنصر في المصفوفة size [i] وفي السطر الثامن عشر يتم حساب آخر عنصر في المصفوفة size وهو الذي يحوي متوسط رواتب جميع الموظفين عبر الجمع التراكمي ثم في السطر 19 وهو سطر مهم جداً إذ أنه يقوم بإفراغ محتويات المتغير sum ؛ إذا لم تضف هذا السطر إلى برنامج فسيحسب البرنامج رواتب موظفي القسم الثاني زائداً عليها رواتب القسم السابق ؛ لذلك يجب عليك إفراغ محتويات المتغير sum.

البحث المتتالي:

طريقة البحث المتتالي هي إحدى الطرق المعتمدة في البحث سوف نقوم من خلال هذا القسم معرفة ما تتضمنه هذه الطريقة.

أفضل طريقة لكي تتناول هذا الموضوع هو وضعه عبر كود

مثال كودي:

أكتب برنامج يطلب البرنامج فيه من المستخدم إدخال درجة أحد الطلاب ثم يقوم البرنامج بالبحث داخل مضفوفة مخزنة مسبقاً في البرنامج عن رقم هذا الطالب الذي أحرز النتيجة المدخلة ويقوم بإدخال رقم الطالب في المصفوفة؟
ومصفوفة درجات الطلاب هي كالتالي:

int mark[10] = { 100,90,69,45,87,52,99,98,99,88} ;

حل المثال:

حل هذا السؤال بسيط لنحدد أولاً المدخلات؛ أول مدخل بالطبع هي مصفوفة الدرجات (ليست مدخل بالتحديد ولكن يكفي أنها مهيئة داخل البرنامج) ؛ ثم يطلب البرنامج من المستخدم إدخال الدرجة المراد البحث عنها وهذا المدخل الثاني ؛ بعد ذلك يقوم البرنامج بمقارنة جميع درجات الطلاب مع الرقم المدخل فإذا وجده يقوم بطباعة رقم الطالب وإذا لم يجده يخبر المستخدم أنه لا وجود لهذه الدرجة. كما تلاحظ فسنضطر لإستخدام دالة تكرارية للبحث والأمر الثاني دالة للقرارات لتقرير إذا كان الرقم المدخل موجوداً أو لا.
أيضاً سنحتاج متغير يستطيع تقرير إذا ما كان البرنامج وجد القيمة أو لا؛ وسنسميه found بحيث أنه إذا أرجع قيمة تساوي الصفر فإن النتيجة غير موجودة وإذا أرجع قيمة تساوي الواحد فإن النتيجة موجودة
والكود سيكون كما يلي:

CODE

```
1       #include <iostream.h>
2       main( )
3       {
4         int mark[10] = { 100,90,65,45,87,52,99,97,87,98} , found ,
index ;
5       int search;
6
7      cout << "Please enter the mark you want it\t" << endl;
8      cin >> search;
```

```
9
10    for ( index=0;index<10;index++)
11    {
12    if (mark[index] == search)
13    {
14    found=1;
15    break;
16    }
17    else
18    found=0 ;
19
20    }
21   if (found=1)
22   cout << "The number of student is:"  << index++ ;
23   else
24   cout << "No Body has this mark" ;
25   return 0;
26   }
```

دعنا الآن نقوم بشرح الكود السابق؛ كما ترى فلقد وضعنا في السطر العاشر دالة تكرارية وظيفة هذه الدالة هي التحرك من أول عنصر إلى آخر عنصر في المصفوفة ؛ كل عنصر من العناصر سيقوم بمقارنتها مع الرقم الذي أدخله المستخدم search وإذا وجد البرنامج أن المقارنة نجحت في السطر 12 سيقوم بإعطاء المتغير found القيمة 1 ؛ ثم يخرج من التكرار for نهائياً وينتقل إلى السطر 21 ؛ أما إذا لم تنجح المقارنة في السطر 12 فيقوم البرنامج بالإنتقال إلى السطر 17 ويقوم بإسناد القيمة 0 إلى المتغير found ثم يرجع إلى قمة التكرار ويقوم بمقارنة عنصر آخر من المصفوفة فإذا لم يجد فكما تعلم أن قيمة found ستكون صفر ؛ نعود إلى السطر 21 في حال كانت found تساوي القيمة 1 فسينفذ البرنامج السطر 22 وإذا وجد البرنامج أن قيمة found هي صفر فإنه ينفذ السطر 24 ؛ كما تلاحظ عند طباعة رقم المصفوفة في السطر الـ 23 فإن البرنامج يضيف واحد إلى الرقم الأساسي وأعتقد أنك تعرف لماذا!.
الطريقة السابقة هي طريقة البحث المتسلسل أو المتتالي.

تصنيف الفقاعات Bubble Sorting :

تعتبر هذه الطريقة هي طريقة فرز وتصنيف ، وقد تتساءل عن فائدة التصنيف أو الفرز والذي يعني ترتيب البيانات وفق ترتيب معين ، الفائدة الكبرى هو تسهيل عملية البحث على الحاسب وبالتالي القدرة على التعامل مع كثير من البيانات بكفاءة كما أن هذا يمهد لاعتماد طريقة البحث الثنائي والتي هي أفضل وأسرع بكثير من طريقة البحث المتسلسل أو المتتالي ، سنتعرض في هذه الفقرة على إحدى الخوارزميات وهي

خوارزمية تصنيف الفقاعات، هذا أحد الامثلة التي حصلت عليها من أحد الكتب[1] يبين لك كيف تنظيم المعلومات بواسطة تصنيف الفقاعات:

عناصر المصفوفة التي نود ترتيبها أو فرزها

50
32
93
2
74

الخطوة الاولى يقارن البرنامج بين العنصر الاول والثاني. ولأن 32 هي أصغر من 50 فإنه يبادل بين أمكنتهم

32
50
93
2
74

من ضمن الخطوة الاولى يقارن البرنامج بين العنصر الأول والثالث ولأن 32 أصغر من 93 فلا يفعل شيء، الآن سيقارن بين العنصر الاول والعنصر الرابع وسيبادل بين أماكنهم

2
50
93
32
74

من ضمن الخطوة الأولى أيضاً يقارن البرنامج بين العنصر الأول والعنصر الأخير 2 و 74 ولن يقوم بأي حركة وبالتالي تنتهي الخطوة الأولى

الخطوة الثانية يقارن فيها البرنامج العنصر الثاني ببقية العناصر، وسيقارن الآن بين العنصر 50 و93 وسيتركهم وسيقوم بعد ذلك بتبديل مكان العنصر الثاني بالعنصر الثالث وسيبدل أماكنهم

2
32
93
50
74

من ضمن الخطوة الثانية يقارن البرنامج بين بين العنصر الثاني 32 والأخير74 ولن يقوم بتحريكهم وبالتالي تنتهي الخطوة الثانية

الخطوة الثالثة يقارن فيها البرنامج العنصر الثالث ببقية العناصر ، وسيقارن أولا الرقم 93 بالرقم 50 وسيقوم بتحريك القيمتين وتبديل أماكنهم

2
32
50
93
74

من ضمن الخطوة الثالثة يقارن البرنامج القيمة 50 بالقيمة 74 ولن يقوم بتبديل الأماكن.

ينتقل البرنامج إلى الخطوة الرابعة وهي آخر خطوة وفيها سيقارن العنصر الرابع بالعنصر الأخير ولن يقوم بتبديل الأماكن وهكذا يصبح شكل المصفوفة مرتباً

الآن سنقوم بجعل هذه الخوارزمية إلى كود وأول ما نود القيام به هو معرفة كم حلقة تكرارية نقوم بها والجواب هو حلقتين اثنتين ، فكما ترى فإن البرنامج يتحرك حول العناصر وهذه الحلقة الأولى ثم يقارن هذه العناصر

[1] اسم الكتاب هو C Languge للمؤلف جريج بيري (المثال مترجم من قبلي)

بالعناصر التي تليها وهذه هي الحلقة الثانية ، الآن علينا معرفة كم عدد المرات التي تتحركها الحلقات والجواب بسيط في الحلقة الأولى تحرك البرنامج في مثالنا السابق أربع خطوات أي أن الحلقة الأولى تتحرك (عدد عناصر المصفوفة – 1) أما الحلقة الثانية فهي تتحرك ببساطة (عدد عناصر المصفوفة – رقم الخطوة التي وصلت إليها الحلقة الثانية).
الآن سنقوم بكتابة الكود الذي ينظم هذه العملية ، وهو كالتالي:

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.     int array[5]={50,32,93,2,74};
7.     int sure=0;
8.     int x=0;
9.     cout << "Here is the Array befor sorted\n";
10.         for (int j=0;j<5;j++)
11.                 cout << array[j] << endl;
12.
13.         for (int i=0;i<5-1;i++) {
14.                 sure=0;
15.                 for (int j=i; j<5;j++) {
16.                         if (array[j] <array[i]) {
17.                                 x=array[j];
18.                                 array[j]=array[i];
19.                                 array[i]=x;
20.                                 sure=1;
21.                         }
22.                 }
23.                 if (sure ==0) break;
24.         }
25.
26.         cout << "Here is the Array after sorted\n";
27.         for (i=0;i<5;i++)
28.                 cout << array[i] << endl;
29.
30.         return 0;
31.     }
```

سأترك لك شرح الكود الحالي وفي حال عدم فهمك له فعد للكلام عن تصنيف الفقاعات النظري وحاول أن تفهم المثال الذي جلبته إليه لفهم خوارزمية تصنيف الفقاعات.
انتهينا الآن من الكلام عن أساسيات المصفوفات وموضوع المصفوفات يعتبر مقدمة صغيرة للغاية عن مواضيع كبيرة مثل المكدسات والأشجار والقوائم المترابطة.. إلخ ، وسننتقل الآن إلى موضوع السلاسل والتي هي في جانب من جوانبها عبارة عن مصفوفة حرفية.

السلاسل (المصفوفات الحرفية)

مقدمة:

سنبدأ بداية من الكلام الذي قلناه سابقاً عن المصفوفات ، أنت تعلم أنه لا يمكنك تخزين أي كلمة في أي متغير حرفي لأن المتغير char عبارة عن بايت واحد فقط وبالتالي فلن يخزن لك إلا حرف واحد فحسب ، سنستغل الآن فائدة المصفوفات وسنقوم بتخزين كلمة كاملة في مصفوفة حرفية:

```
char word[]= { 'P','r','g','r','a','m','\0'};
```

لقد قمنا بتخزين الكلمة Program في المصفوفة word ، أما عن الحرف الأخير وهو \0 فهذا الحرف مهم للغاية وهو يعلم المترجم بانتهاء السلسلة الحرفية ، لو افترضنا أنك لم تقم بكتابة ذلك الحرف ، فعندما تقوم بكتابة السطر التالي لطباعة السلسلة:

```
cout << word << endl;
```

فستظهر لك أحرف غريبة لذلك احرص على إعلام المترجم بنهاية السلسة.
يعتبر الأسلوب السابق أسلوباً غير عملي وممل للغاية وخاصة وجود الحرف الأخير ، لذلك فهناك طريقة أسهل للإعلان عن المصفوفات الحرفية (السلاسل) وهي هكذا:

```
char word[]= "Hellow C++";
```

وهكذا فلن تحتاج للفصل بين الحروف ولا إلى حرف الإنهاء الأخير ، والذي سيقوم المترجم بإضافته نيابة عنك

هناك أمر آخر وهو حجم الكلمة السابقة ، قم بعد الأحرف وستجد أنها 9 أحرف ، ولكن حجم تلك المصفوفة هو 10 بايت والسبب في ذلك هو وجودة مسافة فارغة بين الكلمتين Hellow و C++ والتي تعتبرها السي بلس بلس حرفاً كأي حرف آخر.

إدخال المعلومات في السلاسل:

لنفرض أنك تقوم بكتابة برنامج تطلب فيه من المستخدم كتابة اسمه ، حينها فلربما سيحتوي الكود على هذه الأسطر:

```
	char name[100];
	cin >> name;
```

وبالرغم من صحة الأسطر السابقة ، ولكن ماذا لو قرر المستخدم كتابة اسمه بالكامل أي اسمه واسم أبيه ، نحن لا نناقش هنا حجم المصفوفة فبإمكانك تغييرها متى تشاء ، لنفرض أن المستخدم أدخل اسمه هكذا:

```
Mohamed Abdullah
```

حينها سيقوم الكائن cin بتخزين الكلمة الأولى في المصفوفة ولن يقوم بتخزين الكلمة الثانية أبداً في المصفوفة والسبب في ذلك هو أن الكائن cin يعتبر حرف المسافة الخالية هو حرف إنهاء وبالتالي فإنه ينتهي من القراءة.

لحل هذه المشكلة يوفر لنا الكائن cin تابعاً هو التابع get والذي يقوم بقراءة المسافات الخالية . حينها ستقوم بتعديل الأسطر السابقة لتصبح كالتالي:

```
char name[100];

cin.get (name , 99 );
```

يستقبل التابع وسيطين اثنين هما اسم المصفوفة والقيمة القصوى لعدد الأحرف والسبب في أننا وضعنا الرقم 99 هو للسماح بوجود الحرف الخالي أو حرف الإنهاء ، وهناك وسيط ثالث وهو حرف الإنهاء ، ولا يشترط لك وضعه ولكن عليك أن تعلم أن حرف الإنهاء هو '\n' ، أي إذا قمت بضغط الزر Enter على لوحة المفاتيح فسينتهي البرنامج من قراءة السلسلة التي تكتبها.

التابع getline :

لنفرض أنك ستقوم بكتابة كود يعمل كمحرر نصوص ، فحينها يجب عليك التعامل مع الأحرف '\n' كما رأينا فإن التابع get( ) يقوم بالتعامل مع المسافات ولكن ماذا لو أردت أنت التعامل مع الأسطر وليس الجمل فحسب.
يوفر لك الكائن cin التابع getline الذي يتعامل مع هذه المشكلة. وطريقة عمله هي نفس طريقة عمل التابع get وحتى تجعل التابعين يقومان بحل المشكلة المطروحة (مشكلة الأسطر) فعليك فقط أن تحدد ما هو البارامتر الثالث أو الوسيط الثالث وحينها ستحل المشكلة.

نسخ السلاسل:

توفر لك لغة السي القديمة تابع لنسخ سلسلة إلى سلسلة أخرى وهو التوابع strcpy( ) وطريقة استخدامه بسيطة وهو يستقبل وسيطين اثنين ، الوسيط الأول هو السلسلة المراد النسخ إليها والوسيط الثاني هو السلسة المنسوخة ، انظر إلى المثال الكودي التالي:

CODE

```
1. #include <iostream>
2. #include <cstring>
3. using namespace std;
4.
5. int main()
6. {
7.    char string1[100];
8.    char string2[]= "I am a good programming";
9.
10.        strcpy(string1,string2);
11.
12.        cout << string1 << endl;
13.        cout << string2 << endl;
```

```
14.
15.         return 0;
16.   }
```

فـي السـطر الثـاني قمنـا بتضـمين المكتبـة string القديمـة الخاصـة بلغـة السـي، ولأنها مـن لغـة السـي فلقـد كتبنـا قبلها حـرف c ، لتصـبح هـكذا: cstring ، تحوي هذه المكتبة التابع strcpy ، وكما ترى فلقد قمنا في السـطر العاشر بوضع السلسة string1 كأول وسيط لأنها هـي السلسـة التـي نريـد النسخ إليها أما الوسـيط الثـاني فهـو string2 ، وهـو السلسـلة التـي نريـد نسـخ محتوياتهـا إلـى السلسـلة string1 ؛ فـي السـطرين 12 و 13 ، قمنـا بطباعة محتويات السلسلتين حتى تتأكد من صحة قيام التابع strcpy بعملـه.

المكتبة ctype :

توجد إحدى المكتبات المهـمـة في لغـة السـي القديمـة وهـي المكتبة ctype التـي تقـدم لـك الكثيـر مـن الخـدمات المتنوعـة والتـي قـد تفيـدك أيضـاً فـي المستقبل.

إختبار الحرف:

تستطيع اختبار ما إذا كان المتغير الذي قام المستخدم بادخاله هـو حرف أو لا ووسـيلتك لهـذا هـو التـابع isalpha ، يسـتقبل هـذا التـابع وسـيط واحـد هـو المتغير الحرفي الذي تـود اختباره . انظر إلى هذا المثال:

CODE

```
1. #include <iostream>
2. #include <ctype.h>
3. using namespace std;
4.
5. int main()
6. {
7.
8.    char m='a';
9.    cin >> m;
10.
11.         if (isalpha(m))  cout << " Yes" ;
12.         else cout << "NOOOOOOO";
13.
14.         cout << endl;
15.   return 0;
16.   }
```

الآن فـي حـال مـا إذا قمـت بادخـال عـدد أو أي علامـة أخـرى غيـر الحـروف الانجليزية (صغيـرة أو كبيـرة) فـإن التـابع سـيختبر المتغيـر m وفـي حـال كـان كذلك فسينتقل التنفيذ إلى السطر 12 ، أما إذا كـان حرفـاً فسـيبقى التنفيـذ

في الجملة if . قد لا ترى أي فائدة من هذا التابع ولكن قد يأتي اليوم الذي تستفيد منه ولربما تستفيد منه في إنشاء مشروع آلة حاسبة يفوق الآلة الحاسبة التجارية.

التابعان isupper و islower :

لنفرض أنك تقوم بانشاء برنامج لتسجيل أسماء وتخزينها في أي قاعدة بيانات ، حينها ستضطر إلى التعامل مع الادخالات الخاطئة (لنفرض أن البرنامج باللغة الانجليزية) ماذا لو قام المسجل أو مستخدم البرنامج بجعل أول حرف من اسمه حرفاً صغيراً أو جعل جميع حروف اسمه كبيرة ، بالتالي فإن عليك فعل واحد من اثنين:

- **إنذار المستخدم أنه أخطأ في الإدخال والطلب منه الإعادة .**
- **تصحيح أخطاء المستخدم وإكمال البرنامج كأن شيئاً لم يكن.**

والخيار الثاني هو أفضل ، إلا أن في بعض الحالات عليك التعامل مع جميع الخيارات وقد يكون في أحد الأنظمة ما يجبرك على القيام بالحل الأول ، ففي البرمجة لا يمكنك توقع العوائق التي ستقابلها والتي لن تجتازها إلا إذا كنت عارفاً بأغلب الحلول إن لم يكن كلها.
سنقوم الآن بكتابة مثال يستخدم الحلقة for وستكون هذه الحلقة أبدية ولن يخرج منها المستخدم إلا إذا أدخل الحرف @ ، وهي تقوم باختبار كل حرف يدخله المستخدم ، انظر إلى هذا المثال:

CODE

```
1. #include <iostream>
2. #include <ctype.h>
3. using namespace std;
4.
5. int main()
6. {
7.    for(;;){
8.    char m='a';
9.    cin >> m;
10.         if (m=='@') break;
11.         else if (isupper(m))  cout << "Big char" ;
12.         else if(islower(m)) cout << "Small char";
13.         else cout << "TRY AGINE";
14.
15.          cout << endl;
16.        }
17.   return 0;
18.         }
```

- **يقوم المستخدم بادخال قيمة المتغير m ثم تقوم تفرعات if باختبار هذا المتغير.**

- يقوم السطر 10 باختبار ما إذا كان الحرف هو @ وفي حال كان هكذا فإنه يخرج من الحلقة التكرارية for وبالتالي ينتهي البرنامج .
- السطر 11 يتأكد إن كان الحرف المدخل هو حرف كبير وفي حال كان كذلك فإنه يخبرك المستخدم بذلك.
- السطر 12 يتأكد إن كان الحرف المدخل هو حرف صغير وفي حال كان كذلك فإنه يخبر المستخدم ويطبع رسالة .
- بالنسبة إذا كان المدخل هو حرف آخر غريب أو رقم فإن السطر 13 يتعامل معه .
- مع كل إدخال يدخله المستخدم وبعد اختبار البرنامج له ينتقل التحكم إلى دورة ثانية وإدخال جديد حتى يدخل المستخدم الحرف @ حينها ينتهي البرنامج.

التابعان toupper و tolower :

ربما يكون من الاجدى لك في المثال السابق أن تقوم بتعديل البرنامج حتى يقوم بتغيير الأحرف من كبير إلى صغير والعكس بالعكس ووسيلتك لهذا هما التابعان toupper الذي يقوم بالتحويل إلى أحرف كبيرة والتابع tolower الذي يقوم بالتحويل إلى أحرف صغيرة. سنقوم في هذا المثال برنامج يقوم بتحويل جميع الأحرف التي يكتبها المستخدم عكسها أي الصغيرة إلى كبيرة والكبيرة إلى صغيرة. وفي حال لم يكن هناك أي حرف فإنه يطبع رسالة بهذا الشأن. انظر إلى هذا الكود:

CODE

```
1. #include <iostream>
2. #include <ctype.h>
3. using namespace std;
4.
5. int main()
6. {
7.    for(;;){
8.    char m='a';
9.    cin >> m;
10.
11.         if (m=='@') break;
12.         else if (isupper(m)) {
13.               cout << "Big char\n" << "small char:\t";
14.               m=tolower(m); cout << m;
15.                }
16.         else if(islower(m)) {
17.               cout << "Small char\n" << "big char:\t";
18.               m=toupper(m); cout << m;
19.               }
20.         else cout << "TRY AGINE";
21.
```

```
22.         cout << endl;
23.         }
24.    return 0;
25.    }
```

التابع ( )strcat :

يتبع هذا التابع إلـى المكتبـة cstring ويأخذ كبـارامترات لـه ، وسـيطين اثنـين الأول هـو السلسلة التي تـود إكمالها والثانية هـو السلسـلة التـي تـود أخـذ حروفها وإلحاقها بالسلسلة الأولى. أي أن هذا التـابع يقـوم بـدمج سلسـلتين في سلسلة واحدة.

انظر إلى هذا الكـود الذي يقوم بدمج السلسة الأولى فـي السلسـة الثانيـة، ولاحظ أنه لا يحدث أي شيء للسلسة الثانية المدموجة.

CODE

```
1. #include <iostream>
2. #include <cstring>
3. using namespace std;
4.
5. int main()
6. {
7.    char word1[25]="Java and ";
8.    char word2[10]="C++";
9.
10.         strcat(word1,word2);
11.         cout << "word1:\t" << word1 << endl;
12.         cout << "word2:\t"  << word2 << endl;
13.    return 0;
14.    }
```

ناتج هذا الكـود سيكون كما يلي:

```
word1:      Java and C++
word2:      C++
```

والسبب في دمج السلسة الثانية في السلسلة الأولى هـو السطر 10 حيث التابع strcat ، لاحظ أيضاً أنـه يدمج الوسيط الثاني في الوسـيط الأول ولـيس العـكس.

وهنـاك أيضاً ملاحظة مهـمـة لا يسـتطيع هـذا التـابع تجاوز مشـكلة الكتابـة خارج حدود المصفوفة.

بعض دوال الإدخال والإخراج في لغـة السـي القديمـة:

هـناك أيضاً بعض التوابع التي كانت في لغـة السـي وبالتحـديد فـي المكتبـة stdio ، وهاهي الآن في المكتبة iostream .

التابعان putchar و getchar :

يقوم التابع putchar بعرض حرف وحيد فقط على الشاشة ، وهو يأخذ حرف وحيد فقط لا غير ، أي أنه لا يأخذ حرفان أو ثلاثة بل حرف واحد فقط.
انظر إلى هذا المثال الكودي:

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.    putchar('a');
7.    putchar('\n');
8.    return 0;
9. }
```

أما بالنسبة للتابع getchar فهو مفيد لإدخال حرف وحيد فقط للمتغيرات الحرفية أو السلسلة (ولكن بحلقة for) وإستخدامه أسهل كثيراً من الكائن cin . انظر إلى المثال التالي:

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.    char x;
7.    x=getchar();
8.    putchar(x);
9.    putchar('\n');
10.         return 0;
11.   }
```

انظر إلى كيفية استخدام التابع getchar في السطر 7 .

لاحظ أيضاً هنا أن التابع getchar لن يعمل حتى تضغط على زر الإدخال Enter .

التابع getch( ) :

يتبع هذا التابع المكتبة conio.h ، لذلك احرص على تضمينها في برنامجك قبل استخدام هذا التابع.

يقوم هذا التابع بأخذ محرف واحد وتخزينه في متغير ولا يقوم بإظهاره على الشاشة أي حينما تضغط على أي حرف فإن هذا المتغير لن يقوم بإرسال الحرف الذي أدخلته من لوحة المفاتيح إلى الشاشة.
انظر إلى هذا المثال الكودي:

CODE

```
1. #include <iostream>
2. #include <conio.h>
3. using namespace std;
4.
5. int main()
6. {
7.    char x;
8.    x=getch();
9.    return 0;
10.   }
```

سينتهي هذا البرنامج فوراً حينما تضغط على أي زر في لوحة المفاتيح دون أن يظهر أي شيء على الشاشة.

مثال عملي:
سنقوم الآن بكتابة مثال مسل قليلاً.
لنفرض أننا نطور برنامجاً شخصياً لا يريد صاحبه أن يعرف أحد محتوياته حينها لا بد أن يكون البرنامج معداً بكلمة سر ، وهذا ما سنقوم به الآن.
دعنا نفكر في كيفية تنفيذ هذا البرنامج قليلاً.
يقوم المستخدم بإدخال حرف ولا يظهر على الشاشة بل يظهر حرف النجمة * ، ثم يقارن البرنامج بين كلمة السر المدخلة وكلمة السر المخزنة وحينما تكونان متساويتان يسمح البرنامج لك بالدخول وحينما تكون خاطئة يطلب البرنامج منك الإعادة وإدخال كلمة السر من جديد.
الآن سنستخدم حلقة for الأبدية بالإضافة إلى التوابع السابقة التي تعرفنا عليها قبل قليل.

CODE

```
1. #include <iostream>
2. #include <conio.h>
3. using namespace std;
4.
5. int main()
6. {
7.    int sure=0;
8.    char x[]="book";
9.    char pass[4];
10.         for(;;){
```

```
11.         for(int i=0;i<4;i++){
12.         pass[i]=getch();
13.         putchar('*');
14.         }
15.         for (i=0;i<4;i++){
16.               if (pass[i]==x[i]) sure++;
17.               else break;
18.         }
19.         if (sure == 4){
20.               cout << "\n Password Correct"<< endl;
21.               break;
22.         }
23.         cout << endl;
24.         cout << "False....Try Againe" << endl;
25.   }
26.         return 0;
27.   }
```

- لقد قمنا أولاً بتعريف وإعلان ثلاث متغيرات المتغير الأول هو سلسلة كلمة السر المخزنة والمتغير الثاني هو سلسلة كلمة السر المدخلة أما المتغير الثالث فهو الذي يتأكد أن الكلمتين متساويتان وبالتالي يسمح بالدخول إلى النظام أو البرنامج أو يطلب من المستخدم إعادة الإدخال.
- يدخل البرنامج في السطر 10 في حلقة التكرار for الأبدية.
- يدخل في السطر 11 في حلقة for مختصة بإدخال كلمة السر انظر إلى كيفية الإدخال وإلى ما يظهر في الشاشة.
- في الأسطر من 15 – 18 يقوم البرنامج بالتأكد من تساوي كلمتي السر ، حيث يقارن بين كل حرف وحرف على حدة وفي حال كانت إحدى المقارنات خاطئة يخرج من حلقة for يقوم البرنامج في حال كانت المقارنة صحيحة بزيادة متغير التأكد sure زيادة واحدة.
- إذا كانت المدخلات صحيحة فإن المتغير sure سيصبح يساوي 4 ، وبالتالي يقارن السطر 19 ويتأكد من ذلك وفي حال كان ، يطبع رسالة ترحيبية ثم يخرج من حلقة for الأبدية .
- أما إن لم تكن المدخلات صحيحة فيعود البرنامج إلى التكرار من جديد ويطلب منك إعادة إدخال الكلمة.

وحدة

المؤشرات والإشارات

(صناديق البريد)

بدايـة:

هذا هـو أول موضـوع فـي الكتـاب ؛ بدايـة قويـة للغـاية ... إن سـبب وضعـي فصلاً كاملاً للمؤشرات هـو بسبب أن غالبية من يتعلمون المؤشرات يتناسـون الفائدة منها أو أن بعضهـم لم يحاول فهـم هذا الموضوع فهـماً كـاملاً متكـاملاً ... وهذا ما أحـاول أن أصبو إليه. هذا الفصـل لا يحـاول أن يتعـمق كثيـراً فـي المؤشرات بل سيترك بعض مواضيع المؤشـرات لفصـول أخـرى مـن الكتـاب فالغرض من هذا الفصل هـو إعـطاؤك القدرة على فهـم أفضل للمؤشرات

الذاكرة:

كبدايـة قم بكتابة هذا الكـود

CODE

```
1          // for pointer
2         #include <iostream.h>
3         int main( )
4         {
5          int c=2;
6          cout << &c;
7          return 0;
8          }
```

هذا الكـود بسيط جداً يقوم أولاً بتعريف متغير من نـوع int **ويهيئـه بقيمـة** 2 **... لاحظ في السطر السادس أن مخرج البرنامج ليس** c **وإنما مخرجـه هـو** &c **؛ ماذا تعـني هذه الكلمـة.. إن هـذه العلامــة** & **تعــني إشـارة أو عــنوان ؛ أي أنك تطلب من المتـرجم أن يقـوم بطباعــة إشـارة** c **أو عــنوانها الموجـود فـي الذاكرة ...**
لنأخـذ مثال صـناديق البريـد كمـا تلاحـظ فـإن لكـل صـندوق بريـد عــنوان أو بالأحـرى رقـم صندوق لنفرض أنـه يوجـد فـي هـذا الصــندوق رسـالة هـذه الرسـالة تحــوي العـــدد 2 **، وقـد طلـب منـك طباعــة إشـارة أو عــنوان هـذه الرسالة ؛ أنت لن تطبع محتـوى الرسالة بل سـتطبع رقـم صـندوق البريـد أي عـنوان تلك الرسالة ؛ وهذا ما يقوم بـه البرنـامج السـابق فهــو يطبـع عــنوان المتغير** c **وليس ما يحـويه هذا المتغير ...**

لنعد إلى المثال السابق مرة أخرى وبالتحديد في السطر الخامس .. كما تلاحظ فإنك أعلنت عن متغير هو c وقد تم حجز مقدار له في الذاكرة من نوع int والتي لها حجم محدد من البايت ... الذي فعله المترجم هو أنه قام بإنشاء صندوق بريد ذو عنوان معين هذا الصندوق له حجم معين يستطيع إحتماله وهو 2 بايت ثم يأتي البرنامج برسالة تحوي العدد 2 ويقوم بتخزينها في ذلك الصندوق ... عليك أن تفهم هذه النقطة جيداً .. وهو أنك تستطيع تغيير الرسائل الموجودة في هذا الصندوق من رسالة تحوي العدد 2 إلى رسالة تحوي العدد 6 ؛ لكنك لن تستطيع تغيير عنوان هذا الصندوق ؛ جرب المثال التالي ؛ وسأترك لك مسألة فهمه:

CODE

```
1          // for pointer
2         #include <iostream.h>
3         int main( )
4         {
5          int c=2;
6          cout << &c;
7          c=4
8          cout << &c;
9         return 0;
10          }
```

الآن أتينا إلى نقطة مهمة لنفترض أن لدى البرنامج مساحتين من الذاكرة أول مساحة تسمى stack المساحة الثانية هي heap أي الكومة .. المساحة الأولى تحتوي على عدد صناديق بريد كثيرة جداً إلا أنها ثابتة وإذا انتهت فلن يجد البرنامج مكان آخر لتخزين المتغيرات أما المساحة الثانية heap فهي واسعة جداً إلا أنها فارغة ولا تحوي أي صندوق بريد ولكنها تمتلك ميزة عظيمة وهي أنك تستطيع أنت بنفسك إنشاء ما تريد من صناديق البريد وهناك ميزة ثانية لها أنها أوسع من المساحة الأولى بمئات المرات .... كما رأيت في المثال السابق فنحن لم نتعامل إلا مع المساحةstack فنحن لا نستطيع إنشاء صناديق بريد كما نريد بل يجب أن نلتزم بعدد ثابت من الصناديق نحدده نحن أثناء كتابة البرنامج ولن نستطيع تغييره مهما حاولنا أثناء تنفيذ البرنامج ... ما رأيك الآن أن نتعامل مع المساحة الواسعة والديناميكية heap.

المؤشرات:

من الضروري أن تكون قد فهمت ما كنت أعنيه في مقدمة هذا الفصل حتى تعرف فائدة المؤشرات وخواصها

للإعلان عن أي مؤشر يجب أن يسبق بالمعامل * ثم يكتب إسم المتغير

نوع المؤشر	إسم المؤشر يسبقه المعامل	فاصلة منقوطة
int	* pPointer	;

حسناً الآن ما هو المؤشر ؛ المؤشر هو متغير يشتمل على أحد عناوين الذاكرة... لاحظ أنه يشتمل على أحد عناوين الذاكرة وليس بالتالي قيمة ؛ حتى تفهم ما هو المؤشر فلنعد إلى مثال صناديق البريد ؛ المؤشر يقوم بحجز مكان في الذاكرة (أي صندوق بريد) ثم يشير إلى عنوان هذا الصندوق ... بالتالي لن تستطيع أن تقول:

int *pAge=x;

السطر السابق خطأ ؛ تذكر المؤشر يحمل عناوين ويشير إلى قيمها ولا يحمل القيمة بحد ذاتها؛ وحتى تستطيع إسناد قيمة x إلى المؤشر pAge فعليك أن تكتب التالي:

pAge=&x;

لقد أصبح الإسناد هكذا صحيحاً فكأنك تقول خذ عنوان المتغير x وقم بوضعه في عنوان المؤشر *pAge .. الآن حينما تريد طباعة القيمة الموجودة في المؤشر فإنك تكتب هذا السطر

cout << *pAge;

هذا يعني أنك تقول للمترجم أيها المترجم هل ترى العنوان الذي يشير إليه المؤشر ؛ قم بطباعة القيمة الموجودة في العنوان الذي يشير إليه المؤشر.
بالنسبة لمثالنا السابق (أقصد هنا مثال صناديق البريد) فإن المؤشر هو رقم صندوق البريد المكتوب على الرسالة ؛ والذي تستطيع أنت شطبه وتغييره متى ما أردت بل وحتى تعديل ما هو مكتوب في الرسالة وجعلها تحوي بيانات أكثر وما إلى ذلك أما بالنسبة للإشارة والتي تقابل هنا رقم صندوق البريد فهي ثابتة ولن تتغير
الآن دعنا من هذا الكلام ؛ ودعنا نلقي نظرة فاحصة على هذا الكود:

CODE

```
#include <iostream.h>
 void main ( )
      {
           int p,g;
           int *x;
           p=5;g=7;
           cout << p << "\t" << g << "\n " << &p << "\t" << &g
           << endl;
           x=&p;
           cout << *x << "\t" << x << endl;
           x=&g;
           cout << "\n\n" << *x << "\t" << x;
      }
```

حسناً كما ترى قمنا بتعريف متغيرين p و g ومؤشر واحد هو x قمنا بإسناد القيم للمتغيرين في السطر السادس ثم قمنا في السطر السابع بطباعة قيم المتغيرين وأسفل كل قيمة طبعنا عنوانها في الذاكرة (أو بالاحرى عنوان المتغير الذي يحويها) قمنا في السطر التاسع بجعل المؤشر x يحمل عنوان المتغير p وقمنا بطباعة قيمة المؤشر وعنوان هذا المؤشر في السطر العاشر ؛ الآن لو كنت شديد الملاحظة فستلاحظ أن عنوان المؤشر x هو نفسه عنوان المتغير p ؛ قمنا بعد ذلك في السطر الحادي عشر بجعل المؤشر يحمل عنوان المتغير g وقمنا بطباعة قيمة المؤشر وعنوانه وستلاحظ أيضاً أن عنوان المؤشر هو نفسه عنوان المتغير g.

الآن أعتقد أنك عرفت فائدة المثال السابق .. للمؤشر ميزة عظيمة وهي أنه دائماً يقوم بتغيير عنوانه في الذاكرة (ستتعلم أنه يستطيع تغيير عدد البيانات التي يحويها) بعكس المتغيرات والإشارات .. المتغيرات قيمها متغيرة إلا أن عناوينها ثابتة أما الإشارات فعنوانها ثابت وقيمتها ثابتة ولن يمكنك تغيير قيمة الإشارة بل يجب عليك تهيئتها عند الإعلان عنها أما المؤشر فبإمكانك تغيير عنوانه وقيمته.

لنعد إلى المثال الكودي الأخير لنفرض أني قمت بإضافة هذين السطرين في الكود بين السطر العاشر والحادي عشر(حاول تجريبها بنفسك):

```
*x=8;
cout << p;
```

ستلاحظ أن قيمة p لن تكون نفسها 7 بل ستتغير إلى 8 ؛ مع العلم أننا لم نقوم بأي شيء يغير قيمة P ولكن هل تتذكر السطر التاسع حينما أخبرنا المترجم أن نفس عنوان p هو نفسه عنوان x ؛ من أجل ذلك قام المترجم بوضع قيمة 8 في عنوان المؤشر x الذي هو نفسه عنوان المتغير p ؛ وتذكر أن كل ما سنفعله في المؤشر سيحدث نفس الشيء مع المتغير p إلا إذا وصلنا للسطر الحادي عشر حينما غيرنا عنوان المؤشر من عنوان المتغير P إلى المتغير g.

**حجز الذاكرة للمؤشرات:

هل تتذكر حينما قمنا بتشبيه المؤشر على أنه مثل الرسالة وأن هذه الرسالة تحوي أي عدد من البيانات وأنك تستطيع تكبير حجم هذه الرسالة إلى أي مدى تريده .. عموماً هذا ما ستتعلمه من هذه الفقرة
إذا قمت بكتابة برنامج وكتبت السطر التالي:

```
int *x ;
```

فإن المترجم لن يقوم بحجز مكان في الذاكرة للمتغير x تستطيع أنت فيما بعد أنت تقوم بتعيين عنوان أي متغير آخر لهذا المؤشر .. ولكن ماهي الفائدة من ذلك ؛ فكما تعلم نحن نريد الإستفادة من المؤشرات وليس مجرد المعرفة ؛ إذاً عليك أن تحجز مكان في الذاكرة لهذا المتغير x بحسب ما تريد قم بدارسة المثالين التاليين:

```
1     // for pointer
2    #include <iostream.h>
3    void main( )
4    {
5     int *c
6     *c=50;
7     cout << *c;
8     }
```

أما المثال الثاني:

```
1     // for pointer
2    #include <iostream.h>
3    void main( )
4    {
5     int *c=new int;
6     *c=50;
7     cout << *c;
```

```
8       }
```

كما ترى فإن المثال الأول لن يعمل مهما حاولت بالرغم من أنه لا تعقيد ولا غبار عليه إلا أنه في السطر الخامس من المثال الأول فإنك قمت بالإعلان عن متغير اسمه c يحمل عنوان ؛ لم تقوم بتعيين ما هو هذا العنوان ولا تدري أصلاً أين سيقوم المترجم بوضع الرقم 50 في أي مكان فليس هناك حجز في الذاكرة باسم c من أجل ذلك لن يعمل المثال الأول أما المثال الثاني فسيعمل بالطبع والفرق بينه وبين المثال الاول هو في السطر الخامس حيث استخدمت كلمة جديدة وهي new وهذه الكلمة هي التي تحجز موقع في الذاكرة وهو كما تلاحظ من النوع int لذلك حينما يصل المترجم إلى السطر السادس فسيعلم أين يضع قيمة c.

الآن وبعد أن انتهينا من المثالين السابقين فيجب عليك أن تعلم التالي؛ لا يمكنك تعيين قيمة إلى عنوان فالمتغير c هو عنوان وليس قيمة لكن بإمكانك تعيين عنوان إلى عنوان أو قيمة إلى قيمة ... وحتى تقوم بتعيين قيمة إلى أي مؤشر يجب عليك أن تحجز مكان في الذاكرة لهذا المؤشر وهو كما رأينا بالكلمة new وأن تكتب فيما بعد كلمة new نمط هذا الحجز هل هو int أم غيره مع ملاحظة أنه ليس بإمكانك الإعلان عن مؤشر نمطه int وتحجز له في الذاكرة نمط float.

**الإشارات أو المرجعيات:

الآن سندخل في موضوع شبيه بالمؤشرات وسيمنحك الكثير حينما تبدأ في التعامل مع المؤشرات

ادرس هذا المثال:

```
1        // for reference
2       #include <iostream.h>
3       void main( )
4       {
5        int c;
6        c=50;
7        int &One=c;
8        cout << One;
9         }
```

كما ترى فقد أعلنا عن مرجعية تدعى One ويجب عند الإعلان عن أي مرجعية أن نسبقها بالمعامل & ؛ لاحظ مخرجات البرنامج والتي ستكون نفس قيمة المتغير c

عليك أن تعرف أن المرجعيات لا يمكن الإعلان عنها دون تهيئة وهي في الأساس تستعمل كأسماء مستعارة للهدف ؛ انظر لهذا المثال وادرسه:

```
1        // for reference
2       #include <iostream.h>
3       void main( )
4       {
5        int c;
6        c=50;
7        int &One=c;
8        cout << One << endl;
9         c=80;
10        cout << One << endl;
11         c=500;
```

```
12	cout << One<< endl;
13	}
```

كما تلاحظ فلقد قمنا بتعيين قيم جديدة للمتغير c وفي كل مرة يطبع البرنامج المرجعية One إلا وأنه حسب السطر السابع فإن المتغير c معين إلى المرجعية One وبالتالي فأي عملية على المتغير c تعني أنها ستجري حتماً على المرجعية One.
هذا هو كل موضوع الإشارات ؛ أما عن طريقة حجز الذاكرة لهذه المرجعيات أو الإشارات فهي نفس طريقة حجز الذاكرة للمؤشرات عن طريق الكلمة الدليلية new.
وهذه طريقة حجز الذاكرة للإشارة.

```
char &Refrence= *(new char);
Refrence = 'x';
```

ملاحظات ضرورية حول المرجعيات:
حينما تعلن عن إشارة وتقوم بتهيئتها لتصبح اسم بديل عن الهدف فلن يمكنك تغيير قيمتها مرة أخرى ولن تستطيع تغيير مرجعيتها مهما حاولت ؛ وأي محاولة لتغيير قيمتها فإنها في الحقيقة ستغير من قيمة مرجعيتها أي المتغير الذي تشير إليه انظر لهذا المثال:

```
1	int x=5;int &refrence=x;
2	int y=6; refrence=y;
```

السطران السابقان صحيحان فكما ترى في السطر الأول أسندنا قيمة المرجعية إلى متغير x ثم في السطر الثاني أسندنا قيمة المتغير y إلى المرجعية الذي سيحدث في الحقيقة أننا أسندنا قيمة المتغير y إلى قيمة x أي أن قيمة x الحالية أصبحت 6 .
الآن لاحظ المثال التالي:

```
int &Refrence= *(new int);
Refrence =7;
Refrence =8;
```

الآن الأسطر الثلاث السابقة صحيحة فكما ترى في السطر الأول قمنا بحجز ذاكرة للإشارة في السطر الثاني قمنا بتعيين قيمة لها وفي السطر الثالث قمنا بتعيين قيمة أخرى لها ؛ الآن دعنا نكمل المثال السابق ونضيف إليه الأسطر التالية:

```
int x=99;
&Refrence=x;
```

الآن السطر الأخير غير صحيح لأنك قمت بإعادة تعيين جديد للإشارة في السطر الأول الذي يحوي الإعلان عن المرجعية قمت بتهيئتها بمكان جديد في الذاكرة لا علاقة له بالتأكيد بأي عنوان متغير آخر .

تحرير الذاكرة:

كما تعلمنا فإنك حينما تقوم بإنشاء مؤشر وحجز مكان له في الذاكرة ، مالذي سيحدث لهذا المؤشر .. الذي سيحدث لهذا المؤشر أنه سيبقى موجوداً ولن يلغى من مكانه حتى تقوم أنت بإلغائه .. فكما قلنا سابقاً أن المؤشرات تمنحك الحرية المطلقة للتعامل مع الذاكرة سواء من ناحية التخزين أو الإلغاء ؛ ولكن لهذه الميزة ثمنها وصدقني أن الثمن باهظ للغاية

... عموماً حتى تقوم بإلغاء أي مؤشر من مكانه؛ فبإمكانك كتابة الكلمة الدليلية delete قبل اسم المؤشر المراد حذفه أو حتى المرجعية مع مراعاة عدم كتابة معامل المؤشر أو المرجعية ..... مثلاً لنفرض أنك أنشأت مصفوفة مؤشرات هكذا:

float *number [100] [100];

كما تلاحظ تحتوي هذه المصفوفة على أكثر من عشرة آلاف عنصر يستخدمون 40 ألف بايت من الذاكرة وهو رقم ضخم جداً بالتالي فإن عليك حذف هذه المصفوفة فور الانتهاء منها لتحرير الذاكرة من هذا العبء الثقيل جداً.

سنقوم الآن بدراسة هذا المثال:

```
1	#include <iostream.h>
2	main ( )
3	{
4	int *pPointer= new int;
5	*pPointer = 4;
6	delete pPointer;
7	return 0;
8	}
```

قد تتساءل عن الفائدة المرجوة من تحرير الذاكرة حالياً ؛ لكن تذكر هذا الأمر جيداً حاول دائماً أن تلغي الذاكرة بعد الانتهاء منها ؛ ولا تلعب بهذا الأمر ؛ لا تطلب من البرنامج طباعة المؤشر في المثال السابق بعد تحرير الذاكرة.. صحيح أنه سيطلب العدد المطلوب ؛ لكن الأمر كارثي حينما تتعامل مع المشروعات الضخمة أو المتوسطة ... وحينما أقول أنه خطير فذلك لأن المترجم لا يكشف عن هذا النوع من الأخطاء .. المترجم لا يكشف عن تسرب الذاكرة أو عن قيامك بعمليات على مؤشر تم حذفه .. كل هذه الأخطاء ستظهر عند تنفيذ البرنامج وهو ليس أمراً حسناً كما تعلم ؛ فلن تدري أين هو الخطأ ... لذلك إلتزم بكتابة برامج آمنة وليس برامج خطيرة .. سأتناول موضوع خطورة الذاكرة في الجزء الاخير من هذا الوحدة.

المؤشرات	المرجعيات	العناوين	المتغيرات
تحمل عناوين متغيرة	تحمل عناوين ثابتة لا تتغير	بإمكانها حمل ما تريد لأنها الذاكرة	تحمل قيم فحسب
حجمها متغير	حجمها ثابت ويحدد عند تهيئتها	حجمها هو الذاكرة نفسها	حجمها ثابت ولا يمكن تغييره
معامل المؤشر *	معامل المرجعية &	معامل العنوان & يختلف عن معامل المرجعية	ليس له معامل
خطورتها كبيرة جداً	خطورة أقل من المؤشرات	هي سبب كل الخطورة لأنها أساس الجميع	الأكثر أماناً
مرنة جداً وتمنحك تحكم أكثر في برنامجك	أقل مرونة ؛ لا يمكن إعادة تعيينها	هي التي تمنح المؤشرات والمرجعيات والمتغيرات المرونة	ثابتة ليست مرنة بتاتاً
تستطيع إلغاؤها أثناء تنفيذ البرنامج	تستطيع إلغاؤها	لا يمكنك إلغاؤها إلا بعد إنتهاء تنفيذ البرنامج	لا يمكنك إلغاؤها إلا بعد الانتهاء من تنفيذ البرنامج

**الجزء الثاني

فوائد المؤشرات والمرجعيات:

بدايـة:

الآن سـنأتي فـي الجزء الثـاني إلـى موضـوع التطبيـق العـملي للمؤشـرات والمرجعيات ، في الجزء الأول تعـلمت ماهيـة المؤشـرات والمرجعيات والفـرق بينها وبين العـناوين والمتغيـرات ؛ يجب أن تفهـم الجـزء السـابق فهـو مهـم وضروري جـداً لفهـم بقية هذا الفصل وفصول أخـرى من هذا الكتـاب.

مميزات المؤشـرات:

أحد أهـم مميزات المؤشرات أنها تتعامـل مـع الـذاكرة heap **؛ وأنها متغيـرات لكن بدلاً من أن تحـمل قـيم فإنها تحـمل عـناوين. فـي الـذاكرة وأنـك أيضـاً تستطيع تحـديد شكلها وحجمها في الذاكرة وهـي أيضاً متغيرة وليست ثابتـة ؛ أي ان المستخـدم يستطيع تغيير حجمهـا متى ما أراد أثنـاء تنفيـذ البرنـامج ، والمرجعيات في الأساس تمنحـك أغلب ميزات المؤشـرات.**

المـيزة الأولـى:
تحـمل عـناوين وليس قيـم
(المؤشـرات والمرجعيات والتوابع)

تستفيد التوابع من هذه الميزة فائدة عـظيمـة ، انتظر حتـى نصـل إلـى وحـدة التوابع وسنتعرض لها بالتفصيل.

الميزة الثانيـة:
حجـم المؤشـرات غير ثابت
(المؤشـرات والمصفوفات)

سندخل الآن في تطبيق جـديد ؛ هـل تتـذكر المصـفوفات .. تعلـم أن حجمهـا ثابت دائماً ولا يمكن تغييره مهـما حاولـت فمـثلاً تعلـم أنـت أن السـطر التـالي خاطـيء تمـاماً.

```
int Array [i] [j];
```

حيث i **و** j **أعـداد يدخلها المستخدم في وقت سابق من البرنامج.**
الآن ما رأيك أن نتعلم كيف ننشأ مصفوفة متغيـرة الحجـم وليسـت ثابتـة كمـا في المثال السـابق ... سنقوم أولاً بكتابة السطر القادم:

```
int *Array = new int [i];
```

حيث i **عدد يدخله المستخدم.**
هل تعـلم مالذي سيفعله المترجم حينما يصل إلى السطر السابق.. سيقوم بإنشاء متغير اسمـه Array **ويحجز له في الذاكره ليس عدد صحيح واحـد كمـا في الأحـوال العاديـة بل أعـداد صحيحـة بمثل ما هـو مدخل في العدد** i **فمثلاً لو كـان** i=6 **فسيحجز المترجم ستة أعـداد في الذاكرة للمتغير** Array **حسـناً الآن بإمكـانك إنشاء مصفوفة متغيرة الحجم ؛ ادرس المثال التالي:**

```
1    #include <iostream.h>
2    void main( )
3     {
```

```
4       int  i;
5       cin >>  i;
6       int Array=new int [i];
7        for (int j=0;j<i; j++)
8       cin >> Array[j];
9       }
```

المثال السابق سيعمل دون أية مشاكل ولن يعترض المترجم عليه كما ترى في السطر السادس فسيعمل المترجم على حجز مصفوفة كاملة عدد عناصرها i للمؤشر Array ثم يدخل المستخدم عناصر المصفوفة عبر دوارة for في السطرين السابع والثامن.

الآن نريد أن نقوم بإنشاء مصفوفة متغيرة الحجم لكن هذه المرة ببعدين.

ما رأيك أن نقوم بالإعلان عن مؤشر يشير إلى مؤشر ، كما في السطر التالي:

```
int **pArray;
```

دعنا الآن نقوم بحجز الذاكرة لهذا المؤشر حيث سنحجز له مصفوفة عدد عناصرها i سنكتب السطر التالي:

```
int **pArray= new int *[i];
```

كما قلنا أن هذا المتغير pArray عبارة عن مؤشر يشير إلى مؤشر بالتالي فعندما نحجز له في الذاكرة فسنحجز له مؤشرات لأنه يشير إلى مؤشر وقد حجزنا له مصفوفة كاملة من المؤشرات يبلغ عددها i الآن نريد أن نحجز لهذه المؤشرات مصفوفة أخرى لكل مؤشر فماذا سنكتب ؛ سنكتب الأسطر التالية:

```
for (int k=0;k < i; k++)
Array[k]= new int[j];
```

الآن حجزنا لكل مؤشر مصفوفة كاملة كما في السطر الثاني .. كل الذي عملناه سابقاً هو أننا انشأنا مصفوفة ثنائية متغيرة الحجم.

```
1       #include <iostream.h>
2       void main ( )
3       {
4       int i,j;
5       cin >> i >> j;
6       int **Array=new int *[i] ;
7       for (int k=0 ; k< i ; k++)
8       Array[k]=new int[j];
9       for (k=0 ; k< i ; k++)
10          for (int kk=0; kk< j ; kk++)
11             cin >> Array [k] [kk];
12      }
```

سنقوم الآن بتناول موضوع المؤشر void والمؤشرات الهائمة أو الطائشة بالإضافة إلى المؤشرات الثابتة بالإضافة إلى كلمة بشأن خطورة المؤشرات. وكل هذا في الجزء الثالث من هذه الوحدة.

الجزء الثالث
بقية مواضيع أساسيات المؤشرات والعلاقة بين المؤشرات والمصفوفات:

المؤشرات الهائمة أو الطائشة(stray Pointer):

لن أخوض طويلاً في هذا الموضوع ؛ ولم أضع هذه الفقرة إلا لأنبه على أساليب البرمجة الآمنة ، ينشأ المؤشر الهائم حينما تقوم بالإعلان عن أحد المؤشرات دون أن تقوم بتهيئته ؛ تأكد يجب تهيئة جميع المؤشرات بقيمة أو بعنوان ؛ احذر من عدم تهيئتها ؛ أيضاً لا تنس أنك حينما تلغي أي مؤشر فقم بإسناد القيمة صفر إليه فوراً ؛ فمن المعروف أنك حينما تلغي أي مؤشر فإنه سيبقى يشير إلى منطقة الذاكرة السابقة ، ومكمن الخطورة هنا ، أن المترجم ربما سيحجز مكان لمتغير جديد في تلك المنطقة وتصور مالذي يحدث ، إن الذي سيحدث هو أن يتوقف البرنامج أو أن تحدث له حالة من الجنون فينطلق بلا توقف (قد يوقف نظام التشغيل) ؛ ربما تتعجب من الذي أقوله وتصفني بأني مبالغ ؛ لكن حينما تكتب كوداً طويلاً يتعدي الألف أو حتى المائة سطر وتتعب عليه ثم عند تنفيذ البرنامج يتوقف بلا سبب وتظل تبحث عن هذا الخطأ السخيف (الذي لا ينبهك المترجم عنه) ؛ سيجعلك تشك في أن هناك أخطاء منطقية في البرنامج أو أن الكمبيوتر قد انتهى زمنه مما يجبرك إما أن تترك البرنامج أو أن تعيد كتابته ... لذلك فمن الأسلم لك أن تلتزم بمبادىء البرمجة الآمنة في كل شيء حتى تقاليد التسمية التي تتبعها لمتغيراتك ومؤشراتك.

المؤشرات الثابتة:

لن أتعرض حالياً لهذا الموضوع بشكل دقيق بل سأتركه حينما نصل إلى التطبيقات الفعلية للبرمجة الكائنية .. ولكن من الضروري أن تفهم ماهي المؤشرات الثابتة.
حينما تستخدم الكلمة الدليلية const في أي شيء فإنها تعني ثابت ؛ والتي تخبر المترجم أن لا يغير من قيمة هذا المتغير أو المؤشر الثابت وبالتالي فحينما تغير من قيمة هذا الثابت فسيرسل المترجم رسالة خطأ ؛ الآن أدرس الأمثلة التالية:

```
const int *pFirst;
int *const pSecond;
const int *const pThird;
```

كما ترى في السطر الأول ؛ المؤشر لا يمكن تغيير القيمة التي يشير إليها ؛ من الممكن أن نغير من عنوانه.
في السطر الثاني: من الممكن تغيير المتغير لكن عنوان الذاكرة الذي يشير إليه المؤشر لا يمكن تغييره.
في السطر الثالث لا يمكن تغيير المتغير ولا العنوان الذي يشير إليه المؤشر.

المؤشر void :

هناك أيضاً بعض الخواص في المؤشرات ألا وهي المؤشر void ، بإمكانك أن تقوم بالإعلان عن مؤشر من النوع void ، هكذا:

```
void *pointer;
```

المؤشـرات والمصفوفات:

العلاقـة بـين المصـفوفات والمؤشـرات فـي لغــة الســي بلـس بلـس علاقـة حميمة للغاية بل إن المصفوفات تعتبر قريبة جداً للمؤشرات بشكل لا يصدق.
لو افترضنا أن لديك هذه المصفوفة:

```
int array[10];
```

ولنفرض أنك قمت بهذه العـملية:

```
int *p= & array[0];
```

فحينها سيشير المؤشر pArray **إلى أول عـنصر من المصفوفة.**
وكما تعلمنـا فـإن المصـفوفة عبـارة عــن بيانـات متجـاورة مـع بعضـها الـبعض وبالتالي فإن السطر التالي صحيح:

```
cout << *(p+1);
```

يقـوم هـذا السـطر بطباعــة القيمـة التـي فـي منطقـة الـذاكرة التـي بجانـب المؤشر p **والتي هي العـنصر الثاني من المصفوفة ، وهكذا فبإمكانك طباعة جميع عـناصر المصفوفة بتلك الطريقة.**

التوابع

Function

بدايـة:

لقد تقدمنا كثيراً في السـي بلس بلس بعـد مواضيع المؤشرات والمصـفوفات وربما لم يبقـى لنا سـوى عــدة مواضـيع حتـى ننتقـل إلـى مرحلـة البرمجـة الكائنية وأحد أهـم هذه المواضيع هـي التوابع.

تقوم البرمجـة الهيكلية علـى عـدة توابـع بـدلاً مـن تـابع واحد هــو main( ) **؛ وبإمكـانك بعد هذا الموضوع تجزئـة برنامجك إلـى عــدة توابـع كـل تـابع منها يقوم بوظيفـة محددة ثم يسلمها للآخر بعـد أن يكون قد أتجز ما هـو مطلـوب ؛ ومن الممكـن النظر إلى التوابع على أنها عبارة عـن إتحاد عـدة أوامـر برمجيــة في كتلة واحدة ولهذا الإتحاد وظيفـة معـينة يقوم بأدائها وبالتـالي فسيصـبح بإمكـانك الاستفادة من هذه التوابـع فـي جميـع برامجـك ، فكمـا رأيـت إحـدى توابع المكتبة الرياضيــة** math **وما تقـوم بـه مـن أعــمال ، بإمكــانك أن تقــوم بصنع توابع وضمها في مكتبـة واحــدة ، وأيضــاً فعــن طريـق التوابـع بإمكــانك تجزئـة عـمل برنامجك إلى أجزاء كثيرة وصغيرة للغايـة بدلاً مـن أن تكـون فـي تابع واحد هــو** main **؛ وبصراحــة فإن أغلب البرامج تركت أسلوب التجزئة إلـى توابع وأبدلتـه بتقسيم البرنامج إلي كائنات والكائنــات نفســها تشـتمل علـى توابع كثيرة ، من الضروري للغايـة أن تـدرك أهــمية هـذه الوحـدة إذا مـا أردت التقدم في البرمجـة فأولاً هـي مـدخل إلـى الكائنـات وثانيـاً هــي أحـد أهــم مواضيع لغـة السـي (ليس السـي بلس بلس) والتي لـم تفقـد أهــميتها إلــى الآن.**

بعد هذه المقدمـة البسيطـة سندخل في إختصاص هذه الوحدة.

أساسيات التوابع:

لنلقي نظرة بسيطـة على التابع main( ) **؛ ستجد أنـه مكون من ثلاثـة أشياء كما هـي موضحـة هـنا:**

CODE

```
int  main (  )
{
statment1;
statment2;
statment3;

return  0;
}
```

كما ترى فإن للتابع main( ) **ثلاثـة أجزاء ؛ الأول هــو الرأس والثاني هـو جسم التابع الذي بين القوسين والثالث هـو القيمـة المعادة للتابع وتكتب هكذا:**

return 0

لفهم أفضل لما نقول فسنمضي قدماً في كتابة تابع يقوم بجمع عددين يدخلهما المستخدم.

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int max (int m,int g)
5. {
6.     if (m>g)return m;
7.     else if (m<g)return g;
8.     else return g;
9. }
10.
11.    int main()
12.    {
13.          int num1,num2;
14.          cin>>num1;
15.          cin>>num2;
16.          int max1=max(num1,num2);
17.          cout << max1 << endl;
18.          return 0;
19.    }
```

كما ترى من السطر 4 إلى 9 فلقد قمنا بكتابة تابع أطلقنا عليه إسم max وكما ترى في السطر الرابع فإن التابع يعيد قيمة من النوع int ويستقبل عددين اثنين من النوع int انظر:

القيمة المعادة والوسائط:

لكل تابع قيمة معادة لها نوع محدد يتم إخبار المترجم بنوعها في أول سطر من تعريف التابع هكذا:

int max (int x, int y);

إن الكلمة الاول في الأمر الحالي والتي هي عدد صحيح تعبر عن القيمة المعادة لهذا التابع أما المتغيرات (إكس ؛ وواي) الموجودة بين القوسين في الامر الحالي فهي ما تسمى بالوسائط وهي البيانات الداخلة في التابع لمعالجتها ضمن التابع ، فمثلاً في الكود السابق فإن البيانات الداخلة هي عددين اثنين أدخلهما المستخدم ليقارن التابع بينهما ويعيد الأكبر من بينهما.

في السطر السادس يقارن التابع بين العدد الاول والثاني (أيهما أكبر) ثم يقوم عبر الكلمة المفتاحية return بإعادة العدد الأكبر ، والأمر نفسه ينطبق في السطرين السابع والثامن.

معلومة مهمة:

تذكر أن المترجم حينما يقوم بترجمة أي كود فإنه لا يبدأ الترجمة من أول تابع يصادفه ضمن الكود بل إنه في الحقيقة يبدأ من التابع الرئيسي (الماين) في البرنامج.

بالنسبة للتابع main( ) ؛ فإنه يطلب من المستخدم إدخال عددين اثنين ثم في السطر 16 يقوم بالإعلان عن متغير جديد هو max1 ويقوم بتهيئته بالقيمة المعادة للتابع max ؛ وكما ترى فلقد قمنا بتمرير العددين الذين أدخلهما المستخدم وهما num1 و num2 ، وبالطبع ينتقل التنفيذ إلى التابع max في السطر 4 ، وإذا وصل إلى السطر 9 فإنه يأخذ القيمة المعادة ويهيئ بها المتغير max1 ؛ لعلك الآن تتساءل حول إختلاف الأسماء في المتغيرات بين التابع max والتابع الرئيسي main ؛ في الحقيقة فإنه حينما يصل التنفيذ إلى السطر 16 وبالتحديد لدى هذه الجملة :

```
max (num1 , num2);
```

فإن البرنامج يأخذ معه المتغيرات num1 و num2 ، وينتقل بهما إلى السطر 4 ، وحينما يصل إلى السطر 4 ؛ فإن الترجمة تكاد تكون أشبه بما يلي:

```
m = num1;
g = num2;
```

بمساواة الوسائط الممررة بأول سطر للتابع.

قواعد مجالات الرؤية:

بعد أن انتهينا من أساسيات التوابع فسنمضي قدماً في الحديث عن المتغيرات ولكن هذه المرة من وجهة نظر التوابع ؛ لسنا هنا بصدد الحديث عن الانواع الداخلية لأنماط البيانات مثل int .. وغيرها ، بل حسب قواعد مجالات الرؤية لدى هذه الدالة ؛ عموماً فهناك ثلاثة أنواع للمتغيرات من وجهة نظر التوابع هي كالتالي:

1- المتغيرات الخاصة:
2- المتغيرات العامة:
3- المتغيرات الساكنة:

وسنأتي على كل منها.

المتغيرات الخاصة Local Variables :

هل تتذكر كلامنا السابق في الفصول الأولى من الكتاب حول الكتل ، التابع في الحقيقة ليس إلا كتلة وبالتالي فحينما تقوم بكتابة هذا القوس } فذلك يعني أنك قمت بتدمير جميع المتغيرات التي تم الإعلان عنها بعد قوس الفتح { ، وكما أن الأمر ينطبق على توابع ودوال التكرار فالأمر نفسه هنا بالنسبة للتوابع ، إذا قمت بالتصريح عن أي متغير في أي تابع فحينما ينتهي تنفيذ هذا التابع فإن جميع متغيراته تكون انتهت معه أيضاً وبالتالي فحينما تقوم بإستدعاء نفس التابع مرة أخرى فسيتعامل البرنامج مع المتغيرات وكأنها متغيرات جديدة لم تتم ترجمتها سابقاً ، ومثال الكود السابق هو مثال نموذجي لما نتكلم عنه.

المتغيرات العامة Global Variables :

بعكس النوع السابق فإن المتغيرات العامة هي متغيرات يتم الإعلان عنها خارج أي تابع آخر ، وجميع توابع البرنامج بإمكانها إستخدامها والتعامل معها ،

وحتى نفهم هذا النوع بشكل أفضل ، فدعنا نفرض أن لدينا تابعين اثنين هما:

```
Void test1( )
{
int g=1,k;
}

void test2()
{
int b, g=2;
}
```

كما ترى فإن التابعين الاثنين test1 و test2 يقومان بالإعلان عن متغيرين اثنين إلا أن الأمر الذي نود التأكيد عليه هو أن أحد التابعين لا يستطيع رؤية متغيرات التابع الآخر وبالتالي فلا يستطيع التعامل معها لأنه لا يستطيع رؤيتها ، وكما نرى فإن للتابعين الاثنين متغيرين اثنين لهما نفس الاسم وهو g ولكن ليس لهما نفس مكان الذاكرة وليس لهما نفس القيمة فالمتغير g له نسختين ، كل تابع له نسخة منهما ، الآن لو قمنا بكتابة تعريف لمتغير جديد خارج أي كتلة سواء for أو while أو أي تابع آخر فحينها ستكون متغيرات عامة أي أن جميع الكتل تستطيع رؤيتها ، وبالتالي التعامل معها وكأنها متغيرات خاصة ، إلا أن الفرق هنا هو أن أي تغيير في قيمة هذا المتغير من أي تابع في البرنامج فإن التغيير سيبقى حتى انتهاء البرنامج بشكل نهائي.

المتغيرات الساكنة Static Variables:

المتغيرات الساكنة تأخذ مزايا النوعين السابقين فهي أولاً نفس المتغيرات الخاصة أي أن هناك تابع وحيد يستطيع رؤيتها هو التابع الذي تم الإعلان عنها داخله وثانياً أنها لا تنتهي أو تموت حينما يتم انتهاء تنفيذ التابع في المرة الواحدة فمثلاً لو قمنا بكتابة متغير ساكن ضمن تعريف تابع ما ، فحينما يتم تنفيذ هذا التابع وقام فرضاً بزيادة قيمة المتغير الساكن إلى 2 ، ثم انتهى تنفيذ هذا التابع فإن هذا المتغير الساكن لا ينتهي معه وسيظل محتفظاً بالقيمة 2 حتى يتم استدعاء التابع مرة أخرى وسيجد أن المتغير الساكن أصبح كما هو عليه في المرة السابقة ولن يعود إلى القيمة 0 ؛ باختصار بإمكان تشبيه المتغيرات الساكنة على أنها متغيرات خاصة لا تموت حتى بانتهاء تنفيذ التابع.

وعموماً فإن التصريح عن هذه المتغيرات يتم بالكلمة المفتاحية static كما يرى من هذا السطر:

```
static int number;
```

مثال عملي:

سنقوم الآن بكتابة تابع يقوم بمضاعفة العدد الوسيط إلى ضعفه ومن الممكن أن يكون هذا التابع بداية لك لكي تقوم بإنشاء برنامج حاسبة آلية:

CODE

```
1. #include <iostream>
```

```
2. using namespace std;
3.
4. double binate (float b)
5. {
6.    return b*b;
7. }
8.
9. void main()
10.   {
11.         float a;
12.         cin>> a;
13.         double m=binate(a);
14.         cout << m << endl;
15.   }
```

تـم تعريـف التـابع binate( ) فـي الأسـطر مـن 4 إلـى 7 حيـث يسـتقبل عـدد وسيط واحد وهـو b من النـوع float ويقوم بضربه في نفسـه وإعـادة القيمـة إلى التابع main .

سنتقدم الآن أكثر وسنقوم بكتابة برنـامج أكثـر عـملانيـة وأكثـر فائـدة وهـذه المرة فسنستخدم المؤشرات والمتغيرات العامـة كذلك.
البرنامج الذي نحن بصدده عبارة عـن قواسم عـدد ، المسـتخـدم يـدخل عـدد ما ثم يقوم البرنامج بإنشاء مصفوفـة ثم إسناد كل قاسم مـن هـذه القواسـم إلى عـنصر من عـناصر المصفوفـة ؛ إليك كـود البرنامج:

CODE

```
1.       //   كود يقوم بحساب قواسم أي عدد
2. #include <iostream>
3. using namespace std;
4.
5. float *divides;// المتغيرات العـامـة
6. int times;
7. //////////////////////////////
8. void HowTimes(int x); //النمـاذج المصغرة
9. void pointer();
10.   void TheMain(int x);
11.   ////////////////////////////
12.   void main() //  التابع الرئيسي
13.   {
14.         int a;
```

```
15.         cin>> a;
16.         TheMain(a);
17.         for(int i=0;i<times;i++)
18.         cout <<divides[i]<< endl;
19.         cout << "The Many Of How Number Divides Is:\t"
20.                 <<times
21.                 <<endl;
22.     }
23.     //////////////////////
24.     void pointer( )
25.     {
26.         divides=new float[times];
27.     }
28.     ////////////////////////
29.     void HowTimes(int x)
30.     {
31.         for (int i=1;i<=x;i++)
32.                 if(x%i==0) ++times;
33.     }
34.     ////////////////////
35.     TheMain(int x)
36.     {
37.         HowTimes(x);
38.         pointer();
39.         for (int i=1,int j=0;j<times,i<=x;i++)
40.                 if(x%i==0){
41.                         divides[j]=i;
42.                         j++;}
43.     }
```

لقد احتـوى هذا المثال على مواضيع كثيرة سنقوم بمناقشتها حالاً.

فكرة البرنامج:

لهذا البرنامج متغيران عامين رئيسين هـما:

- **المتغير العام** times : **وهذا المتغير يحسب عـدد الأعــداد التـي تقسـم العـدد المراد إيجاد قواسمـه.**
- **المؤشر** divides: **بعـد أن يحسب البرنـامج عــدد قواسـم العــدد فإنــه يقوم بحجز مصفوفة عدد عـناصرها يساوي قيمــة المتغيـر** times ، **ثـم يقوم البرنامج بتخـزين قواسم العـدد في المصفوفة** divides.

أيضاً فإن لهذا البرنامج ثلاث توابع وهـي كالتالي:

1- التـابع HowTimes(int x) : يسـتقبل هـذا العــدد الـرقم الـذي أدخلـه المستخـدم ويقوم بحساب عـدد قواسمـه ويخزنها فـي المتغيـر العـام times.

2- التـابع pointer( ) : يقـوم هـذا التـابع بحجـز الـذاكرة للمؤشـر divides وهـو يحجز له مصفوفـة حتى يخـزن فيهـا جميـع قواسـم العــدد الـذي أدخله المستخـدم.

3- التابع TheMain(int x) : يعتبر هـذا التـابع هــو أهــم تـابع حيـث يقـوم بإســتقبال الــرقم الــذي أدخلــه المستخـــدم ويــتحكم فــي التــابعين السـابقين ويحسـب قواسم العـدد ويخزنها في مصفوفـة divides.

هذه هـي فكرة هذا البرنامج بشكل عام ولكن هــناك بعـض النقـاط الجــديدة التي يجب التوقف عـندها وشرحها للقارئ العـزيز.

النماذج المصغرة Prototype :

لننظر إلى بداية البرنامج وبالتحـديد في هذا الجزء من الكـود:

```
8.    void HowTimes(int x); //النمـاذج المصغرة
9.    void pointer();
10.   void TheMain(int x);
```

كما ترى فلقد قمنا بكتابة رؤوس التوابـع فقـط وقمنـا بـالتفريق بينهـا بعلامــة الفاصلة المنقوطـة (;) وينصـح دائمـاً فـي أي بـرامج تقـوم بكتابتهـا أن تكتـب النماذج المصغرة لها كما هـو في هذا المثال وللنماذج المصغرة فوائد كثيرة:

1- لنفـرض أن لـديك تـابعين اثنـين ولنفـرض أن التـابع الأول احتـاج إلـى إستدعاء التابع الثاني وفي نفس الوقت فقد يحتاج التـابع الثـاني إلـى إستدعاء التابع الأول أي أن التابعين الاثنين يحتاجان إلى إسـتدعاء كـل واحد منهـما فحينها لن تستطيع كتابة تعريف أحـد التـابعين قبـل الآخـر والنماذج المصغرة تحل هذه المشكلـة.

2- لـن تحتــاج إذا قمـت باستعــمال النمـاذج المصـغرة إلـى كتابـة أسـماء الوسـائط والمتـرجم سـيتجاهل الأسـماء فـي الأسـاس الـذي تحتـاجه فقـط هــو كتابـة نــوع المتغيـرات فلنفـرض أن لـديك تـابع يسـتقبل وسيطين اثنين من النـوع int و float ؛ في حال إذا أردت كتابة النموذج المصغر فإنـه يستحسن أن تكتبه هـكذا:

int test(int , float);

3- أيضاً فإن هـناك فائدة أخرى وهـي أنه عـند تعريف التوابـع تحـت التـابع main فلن تضطر إلى كتابة القيم المعادة للتوابع كمـا هــو ظـاهر لـدى التابع TheMain في السطر 35.

هذه هـي أهـم فوائد النماذج المصغرة.

مشاكل المتغيرات العامـة:

يميل أكثر المبرمجيم المبتدئين إلى إستخدام المتغيرات العامـة فهي تبعــدك كثيراً عـن مشاكل القيم المعادة وتبـادل المعلومـات بـين التوابـع إلا أن هــناك مشاكل كثيرة لها وهــي أنهـا سـتبقى مرئيــة فـي المنـاطق التـي لا تريـدها أيضاً فهي تجعل من عـملية تتبع البرنامج عـملية تكاد تكـون مستحيلــة نظـراً للتعقيد ، ربما تعتبر هـذه المشـاكل هــي التـي جعلـت أول مبـدأ مـن مبـادئ

البرمجة الشيئية يظهر وهو الكبسلة الذي سنتعرض له في الفصول اللاحقة.

تمرير الوسائط بواسطة القيمة:

بالرغم من أننا أشرنا إلى هذا الموضوع إلا أنه لا بد من تناوله في فقرة كاملة ؛ على العموم يوجد نوعان من التمرير بالوسائط إلى التوابع الأخرى:

1- التمرير بواسطة القيمة
2- التمرير بواسطة المرجع

ويعتبر النوع الثاني هو الأفضل والأكثر أماناً إلا أن هناك بعض الحالات التي تضطرك إلى إستخدام النوع الأول.

عموماً تعتبر البارامترات (الوسائط الممررة) متغيرات محلية بالنسبة للتابع الذي مررت إليه ويتم تدميرها عند إنتهاء تنفيذ التابع ، وعموماً فيجب عليك عند كتابة النموذج المصغر للتابع أن تذكر معه نوع البارامترات ولا يشترط ذكر اسمها ، ولكن تذكر أن هذا الأمر خاطيء :

```
int  function (int  x, z);
```

والسبب في ذلك يعود إلى أنك لم تذكر نوع الوسيط الثاني ، صحيح أنه قد يفهم من الامر السابق أنك تقصد أن البارامتر الثاني من النوع int إلا أن المترجم لن يفهم هذا الأمر.

في حال ما إذا كان لديك أكثر من بارامتر فإنك تقوم بالفصل بينها بواسطة الفاصلة العادية وليس الفاصلة المنقوطة ؛ هكذا (,).

القيمة العائدة return Value :

في نهاية كل تابع نجد هذه الكلمة return والتي تحدد ما هي قيمة الإعادة ، انظر إلى هذا الأمر:

```
int  function (int  x,int z);
```

تجد أنه لا بد لهذا التابع أن يعيد قيمة من النوع int ، قد تكون هذه القيمة رقماً أو متغيراً من النوع int ، انظر لهذا الأمر:

```
return  ( 2) ;
```

لا يشترط أن تضع القيمة المعادة بين قوسين ولكن يفضل حتى يصبح الكود أكثر تنظيماً وفهماً ، لربما أنه تعلم أنه بإمكانك إستبدال الرقم 2 بمتغير آخر من النوع int .
ليس ذلك فحسب بل بإمكانك جعل القيمة المعادة تابعاً كاملاً بحد ذاته انظر لهذا المثال:

```
return (function(4));
```

إنه يقوم بإستدعاء تابع إسمه function وسيقوم هذا التابع بإستدعاء نفس التابع وسيستدعي نفسه إلى مالا نهاية مالم تضع للأمر حداً بواسطة القرارات. وسنتناول هذا الإستدعاء المتكرر في موضوع آخر من هذه الوحدة.

أيضاً فإن للقيمة المعادة فائدة كبيرة أخرى وهي أنها تسمح لك بطباعتها دون الحاجة إلى تخزين قيمتها في متغير ما ، فبدلاً من كتابة هذه الأوامر:

```
int number=function (4) ;
cout << number ;
```

كما ترى فلقد قمت بتخزين القيمة المعادة للتابع function في متغير آخر حتى تقوم بطباعتها ، بإمكانك إختصار هذا الأمر إلى هذا السطر:

```
cout << function ( 4 ) ;
```

وسيقوم المترجم بطباعة القيمة المعادة للتابع function ، قد تجد هذه المواضيع سخيفة أو ليس لها من داع ولكن ستسفيد منها كثيراً حينما تصل لمواضيع الكائنات.

ملاحظة مهمة:

تذكر أن أي تابع لم تذكر نوع قيمته المعاده فإنه قيمته المعادة ستكون إفتراضياً من النوع int .

تذكر:

التوابع التي تعيد قيمة من النوع void ليس لها قيمة معادة أي أننا لا نكتب في نهاية التابع return ، تذكر جيداً أن هذه التوابع تعيد قيمة وهي من النوع void حالما نتقدم أكثر ستجد توابع لا تعيد أي قيمة حتى القيمة void .

المعامل (::) :

هناك معامل آخر لم نتعرض له وهو معامل الوصول إلى المتغيرات العامة وهو :: ، انظر إلى هذا المثال:

```
int  a=10;
void function( )
{   int a= 5 }
```

كما تلاحظ فإن هناك متغير خاص أو محلي له اسم a للتابع function ، وهناك أيضاً متغير عام ، السي بلس بلس تسمح لك بفعل ذلك ولكن المتغير العام سيتم إستبعاده أو إخفاءه وستكون الأولوية في التابع function للمتغيرات المحلية وليس للمتغيرات العامة ، وحتى تستطيع الوصول إلى المتغير العام ضمن كتلة التابع function فعليك أن تقوم بكتابة المعامل :: حتى تصل إليه أنظر لهذا الأمر الذي نفترض أنه ضمن كتلة التابع function :

```
cout << ::a ;
```

لن يقوم هذا الأمر بطباعة القيمة الخاصة بالمتغير الخاص بل بالقيمة الخاصة بالمتغير العام لأننا قمنا بكتابة المعامل :: قبل اسم المتغير.

الوسائط الإفتراضية:

أحد أهم أهداف أي برمجة هو إعادة الاستخدام ، أي إعادة استخدام الاكواد السابقة وحتى نصل إلى هذا الهدف فلا بد علينا من جعل استخدام هذه الأكواد السابقة بسيطاً للغاية وبدون أي تعقيد ، انظر مثلاً للكائن cin

وكيف أن إستخدامه بسيط وميسر وأيضاً للدالة printf( ) في لغة السي ومدى سهولتها وهذا أيضاً ما نحاول الوصول إليه من خلال هذا الكتاب.
بإمكاننا تسهيل استخدام أي دالة بواسطة الوسائط الافتراضية (البارامترات الافتراضية) وهذه الأداة تمكننا من تسهيل الكود لدرجة كبيرة ، هل تتذكر التابع getline( ) ، هذا التابع يحتوي على ثلاث بارمترات ، ولكنك تستطيع التعامل معه على أنه يستقبل بارامترين اثنين وتستطيع إذا أردت استخدام ثلاث بارامترات ، نفس الأمر ينطبق هنا ، بإمكاننا إنشاء توابع بتلك الطريقة ووسيلتنا لذلك هي الوسائط الافتراضية.
سنقوم الآن بكتابة مثال كودي وهذه المرة سيقوم هذا المثال بحساب النسبة المئوية ، حيث أنه سيقوم بحساب النسبة من 100 افتراضياً ، وبإمكان المستخدم حساب النسبة من 100 أو أي رقم آخر يريده.

```
1. float rate(float a,float b ,float c=100)
2. {
3. float j=0;
4. j= (a*c)/b;
5. return j;
6. }
```

انظر إلى السطر الأول تجد أن البارامتر الثالث غريب بعض الشيء حيث قمنا باسناد البارامتر إلى القيمة 100 ، وبذلك سيكون بإمكانك استخدام هذه القيمة افتراضياً ، بإمكانك استدعاء هذا التابع بهذا الشكل:

```
rate ( 50 , 100)
```

أو بهذا الشكل إن أردت:

```
rate ( 20, 100 , 1000)
```

والفرق بين الاستدعائين أن البارامتر الثالث للتابع المستدعى الأول هو سيكون افتراضياً بقيمة 100 ، أما التابع الثالث فلقد أراد المستخدم تغيير هذه القيمة وبالتالي فلقد قام البرنامج باستبعاد القيمة الافتراضية ووضع القيمة التي قام المستخدم بوضعها.

سنرى الآن كيف سيكون استخدامنا لهذا التابع في وسط برنامج حقيقي ، عليك أن تعلم أن القيمة الافتراضية لا تكتب أبداً في رأس التابع إلا في النموذج المصغر فقط ، أما تعريف التابع فلا تقم بكتابة القيمة الافتراضية وإلا فإن المترجم سيصدر خطأ ، انظر لهذا المثال ، وكيف تم تطبيق الكلام الحالي:

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4.
5. float rate (float a,float b,float c=100);
6.
7. void main()
```

```
8. {
9.    float i,j,k,avg;
10.         cout << "Please Enter the number?\n";
11.         cin >> i;
12.         cout << "from:\t";
13.         cin >> j;
14.         cout << "the Avrege:";
15.         cin >> avg;
16.
17.         k=rate (i ,j,avg);
18.         cout << endl << k << endl;
19.
20.   }
21.
22.   float rate(float a,float b ,float c)
23.   {
24.         float j=0;
25.         j= (a*c)/b;
26.         return j;
27.   }
```

قارن بين رأس التابع في السطر 22 والنموذج المصغر للتابع في السطر 5 تستنتج أن النموذج المصغر بإمكانه الاحتواء على قيم افتراضية أما رأس التابع أو تعريف التابع فليس بإمكانه الاحتواء على أي قيمة افتراضية.

إعادة أكثر من قيمة بواسطة المؤشرات أو المرجعيات:

الآن سنأتي إلى التطبيق الفعلي للمؤشرات ؛ هل تتذكر التوابع أليس في نهاية كل تابع مالم يكن void العبارة التالية:

return (Value);

حيث value القيمة المعادة.

كما ترى فإن جميع الدوال أو الإجراءات لا تعود إلا بقيمة واحدة ولا تستطيع العودة بأكثر من قيمة ، الآن سنفكر بطريقة تمكننا من جعل التوابع تعود بأكثر من قيمة.

ما رأيك الآن بدلاً من أن نمرر للتوابع القيم أن نمرر لها عناوين تلك القيم ؛ سنكتب برنامج هذا البرنامج يحوي تابعان التابع main وتابع آخر سنطلق عليه plus سيعيد هذا الإجراء قيمتين وسيقوم الإجراء main بطباعتهما وليس التابع plus .

```
1    #include < iostream.h>
2     void  plus (int num1,int num2,int *plus1,int *plus2)
3       {
4       *plus1=num1 + num2;
```

```
5        *plus2=num1*num2;
6         }
7
8        void mian ( )
9         {
10         int num1,num2,plus1,plus2;
11        plus (num1,num2, &plus1 , & plus2);
12        cout << plus1  << endl;
13        cout << plus2 << endl;
14         }
```

الآن وكما ترى فإن قيم plus1 و plus2 ستؤدي المطلوب منها حيث plus1 يجمع عددان و plus2 يضرب عددان بالرغم من أن المعالجة لا تتم في التابع main() بل في التابع plus وكما تلاحظ فإن التابع plus لا يعود أي قيمة لأنه void ؛ كما تلاحظ أعلنا عن عددان مهيئان مسبقاً وعددان لم يهيئا في السطر العاشر ؛ بعد ذلك قمنا بتمرير قيمة العددان num1 و num2 إلى الإجراء plus أما بالنسبة للعددان الآخران فلم نمرر قيمهما بل مررنا عناوين تلك القيم كما هو واضح من السطر الحادي عشر ؛ كما درسنا في هذا الموضوع (موضوع التوابع) أنها تنشأ نسخ من المتغيرات الممررة إليها أما في هذه الحالة فهي لم تقوم بإنشاء نسخة بل أخذت النسخ الأصلية من تلك المتغيرات وهي عناوينها ... الآن يتفرع البرنامج إلى التابع plus والذي عرفناه في السطر الثاني وكما تلاحظ فهو يحتوي عددان من نوع int ومتغيران آخران لكن مؤشرات هذه المرة وليسا متغيرات عادية .. هل تعرف لماذا .. كما تلاحظ فلقد مررنا عناوين تلك المتغيرات ؛ البرنامج الآن بحاجة إلى متغير ليحمل تلك لعناوين وكنا تعلم فإن المؤشر هو متغير يحمل عنوان .. ثم في السطر الرابع والخامس تمت معالجة القيم حيث في السطر الأول جمعنا العددان وفي السطر الخامس ضربنا العددان ثم في السطر السادس عدنا مرة أخرى إلى الإجراء main() ثم في السطران الثاني عشر والثالث عشر قمنا بطباعة النتائج .... وهكذا انتهى البرنامج.

خلاصة هذا الشرح ؛ أنه لكي تجعل التابع يعود بأكثر من قيمة عليك أولاً أن تمرر عناوين أو مرجعيات تلك القيم وليس القيم بحد ذاتها ؛ حينما تقوم بتعريف التابع فإنك تضع في قائمة الوسائط مؤشرات لتلك العناوين المرسلة حتى تستطيع حملها

كما تلاحظ فلقد استخدمنا في المثال السابق المؤشرات ... ما رأيك الآن أن نستخدم بدلاً عن المؤشرات المرجعيات... انظر لهذا المثال وهو نفس المثال السابق لكن هذه المرة نستخدم المرجعيات بدلاً من المؤشرات:

```
1      #include < iostream.h>
2       void  plus (int num1,int num2,int &plus1,int &plus2)
3         {
4         plus1=num1 + num2;
5         plus2=num1*num2;
6          }
7
8        void mian ( )
9         {
10         int num1,num2,plus1,plus2;
11        plus (num1,num2, plus1 ,  plus2);
12        cout << plus1  << endl;
```

```
13        cout << plus2 << endl;
14         }
```

المثال نفس مثال المؤشرات عـدا في السطر الحادي عشر فلقـد تـم إرسـال القيم بدون أي تغيير لها أما تعريف التابع plus في السطر الثاني فلقـد جعلنـا تلك القيم إشارات .

بالرغـم مـن أن المثالين السـابقين سيعـملان بـنفس الجـودة إلا أن المثـال الأخير بإستخدام المرجعيات أقوى مـن المثـال السـابق فهـو لا يجعلـك تفكـر عـند إرسال القيم للإجراء ؛ فـلا يجعلـك تقـول هـل أرسـل عـنوان القيمـة أم القيمـة ؛ وهذا ما تحـاول ++C الوصـول إليـه ؛ خاصـة فـي أمـور البرمجـة الكائنيـة .. عـموماً سنصل إلى جميع نقاط هذه الفوائـد فـي وقـت لاحـق مـن الكتاب

التمرير بالمرجع أفضل من التمرير بواسطة القيمة:

كما تلاحظ فإنـه عـند إرسـال أي قيمـة لأي إجـراء فإنـه فـي الحقيقـة يقـوم بنسخ تلك القيم ووضعها في قائمـة الوسائط الموجودة في إعـلان الإجـراء... بالتالي فإنـك عـندما تمرر عشر قيم إلى أحـد الإجـراءات فكأنـك قمـت بإنشـاء عشرين متغير وليس عشرة ... أما عـندما تمرر عـناوين تلـك القـيم فإنـك فـي الحقيقـة تمرر المتغيرات الأصلية وليس نسخاً عـنها وهذا مـا يـوفر الكثيـر مـن ناحية السـرعـة والأداء وبقية ميزات برنامجـك.

التوابع والمصفوفات:

تعرفنا في الفقرة السابقـة على الفائـدة المرجـوة بـين التوابـع والمؤشـرات ، والآن سنتعرف على كيفية تعامل المصفوفات أو التوابع مع الأخرى.
في الحقيقة فإنه ليس بامكانك إرسـال مصـفوفة دفعـة واحـدة إلا إن كانـت تحتوي على متغير واحد وليس بإمكانك أيضاً جعل التابع يعيد مصفوفة كاملة.
أما عـن كيفيـة انتقال المصفوفات إلى التوابـع فهـي تكـون بـالمرجع حصـراً ، والمترجم هـو بنفسـه سـيقوم بـذلك ، تسـتطيعها إرسـالها بالقيمـة كوسـائط للتوابع ولكن لن يكون بإمكانك سـوى إسـتدعاء التـابع أكثـر مـن مـرة (حسـب عدد عناصر المصـفوفة) أما إذا قمـت بإرسـال المصـفوفة فسـيكون بإمكانـك إستدعاء التابع مرة واحدة فقط لتغيير جميع المصفوفة.
حتى تستطيع جعل تابع من التوابع يستطيع استقبال مصـفوفة كبـارامتر لـه، فعليك أولاً بـإبلاغ التـابع أنـه سيسـتقبل مصـفوفة ، انظـر إلـى أحـد النمـوذج المصغر لتابع يستقبل مصفوفة:

```
void arraysFunction (int [] );
```

لم نقـم في قائمـة الوسائط إلا بذكر نـوع المصـفوفة وكتابـة علامتـي فهـرس المصفوفات ، ثم بعـد ذلك نستطيع التعامل مع المصفوفة وكأنها عــنصر فـي التابع ( )main ، ولا يجب علينا أن نتـدخل فـي أمــور المؤشـرات والمرجعيـات المعقدة ، سنقوم الآن بكتابة كـود يقوم بعـكس عــناصر إحـدى المصـفوفات ، انظر إلى هذا الكـود وحاول فهـمـه قبل قراءة الشرح الموجود تحتـه:

CODE

```
1. #include <iostream>
2. using namespace std;
3.
```

```
4. void arraysf (int [] );
5.
6. int main()
7. {
8.    int array[5]={1,2,3,4,5};
9.    for (int i=0;i<5;i++)
10.             cout << array[i] << endl;
11.         arraysf(array );
12.         for ( i=0;i<5;i++)
13.             cout << array[i] << endl;
14.
15.   return 0;
16.   }
17.
18.   void arraysf(int m[])
19.   {
20.         for (int i=0,int j=5;i<5;i++,j--)
21.             m[i]= j;
22.   }
```

- انظر إلى النموذج المصغر للتابع arraysf ، وهكذا نكون أعلمنا التابع أنه سيستقبل مصفوفة.
- الإعلان عن المصفوفة كان في السطر 8 وهي مكونة من خمسة أرقام من الرقم 1 إلى الرقم 5 .
- السطران 9 و 10 تقوم بطباعة عناصر المصفوفة.
- يقوم السطر 11 باستدعاء التابع arraysf وهو من النوع void ، وسيقوم بمعالجة عناصر المصفوفة بواسطة عناوين الذاكرة، قد تستغرب من هذا الشيء خاصة وأن الكود لم يكتب ليس فيه علامة مرجع ولا مؤشر ولكن هذه الأمور يقوم بها المترجم بنفسه.
- ينتقل التنفيذ إلى السطر 20 ، حيث تقوم الحلقة for بتغيير عناصر المصفوفة عكسياً وحينما ينتهي التنفيذ ينتهي التابع ، لاحظ أنه يرجع قيمة void .
- يعود التنفيذ إلى التابع main( ) ويقوم السطران 12 و 13 بطباعة عناصر المصفوفة بعد تغييرها ، هذه هي نتيجة تنفيذ هذا الكود:

```
1
2
3
4
5
5
4
```

```
3
2
1
```

نقل المصفوفة ذات بعدين إلى التوابع:

بقي أن أشير هنا إلى كيفية نقل مصفوفة ذات بعدين إلى تابع معين، في الحالة الأولى (المصفوفة ذات البعد الأول) لم يكن يشترط ذكر حجم المصفوفة ولكن في هذه الحالة يجب عليك ذكر حجم البعد الثاني للمصفوفة ، وبالتالي فسيكون النموذج المصغر لأي تابع يعالج هذا النوع من المصفوفات هكذا:

```
void  arrayFunction (int [ ]  [ 6 ] );
```

تذكر أن المصفوفات شديدة الشبه جداً بالمؤشرات حتى تفهم عملها وحتى تفهم ما يأتي منها كالقوائم المترابطة والأشجار خاصة في المواضيع المتقدمة، وقد نتناول موضوع القوائم المترابطة و جزءاً من بنى المعطيات في هذا الكتاب.

العودية:

هناك نوع من الخوارزميات يدعي الخوارزميات العودية ، وهذه الخوارزميات لا تعمل إلا بوجود التوابع وربما في بعض الحالات المتغيرات الساكنة ، وحتى تفهمها فهي قريبة جداً من حلقات التكرار إلا أنها أخطر منها حيث أنها في بعض الأحايين تكون غامضة أو شرط توقفها غامضة كحلقات for الأبدية .

لا يمكن فهم العودية إلا من خلال الأمثلة ، لنفرض أن لديك هذا التابع :

```
void Function()
{
      Function( );
}
```

يعتبر هذا المثال مضحكاً للغاية وقد يدمر مشروعك البرمجي حينما تقوم باستدعاء هذا التابع من التابع main( ) فإنه حينما يصل لأول أمر سيقوم باستدعاء نفس التابع وهذا التابع المستدعى سيقوم باستدعاء نفس التابع وستقوم جميع التوابع المستدعاة باستدعاء نفسها إلى مالانهاية ، وقد ينهار برنامجك بسبب ذلك.

إذاً العودية هي أن تقوم الدوال باستدعاء نفسها ، ولكن كما في التكرارات فلا بد لهذا الاستدعاء من نهاية ، وكما يحدث في التكرارات من وجود شرط ، فلا بد في التابع أن يكون هنا من شرط وكما رأيت في الحلقة for والتي تقوم بالعد حتى تصل إلى نقطة معينة ثم تنتهي فإنه بإمكانك إحداث الأمر هنا نفسه في العودية عن طريق المتغيرات الساكنة ، سنقوم الآن بكتابة مثال شبيه بالحلقة for ، وسيقوم هذا التابع الموجود في الكود بطباعة نفسه حسبما تريد من المرات (مثل حلقة for):

CODE

```
1. #include <iostream>
2. using namespace std;
3.
```

```
4. void function (int x);
5.
6. int main()
7. {
8.    int n=0;
9.    cout << "Enter The Number:\t" ;
10.         cin >> n;
11.         function (n);
12.
13.         return 0;
14.   }
15.
16.   void function (int x )
17.   {
18.         static int i=0;
19.         i++;
20.         cout << "Number i=\t" << i << endl;;
21.         if (i==x)
22.               return ;
23.         function(x);
24.   }
```

بالرغم من طول هذا المثال ، إلا أن فهمك لك سيسهل الكثير من الأمور عليك في موضوع العودية (بعض الأشخاص يعتبر صعوبة موضوع العودية مثل صعوبة موضوع المؤشرات) :

- كما ترى في التابع ( )main فإنه طلب البرنامج من المستخدم طباعة الرقم الذي يريد تكراره في السطر 10.
- في السطر 11 تم إستدعاء التابع function وتم تمرير العدد الذي أدخله المستخدم إليه.
- ينتقل التنفيذ إلى السطر 18، حيث تم الإعلان عن متغير ساكن وتمت تهيئته بالعدد 0 (وهذا شبيه بالجزء الأول من حلقة for).
- في السطر 19 تمت زيادة المتغير الساكن i (والذي يعتبر مثل الجزء الثالث من حلقة for).
- في السطر 20 تمت طباعة الرقم الذي وصل إليه التابع (مثل التكرار) .
- في السطر 21 تتم مقارنة الرقم الذي وصل إليه التابع بالرقم الذي أدخله المستخدم في التابع ( )main وفي حالة المساواة تنتهي هذه العودية بالجملة return ، والتي تخرجك نهائياً من هذه العودية (تشبه الجملة break) في حلقات التكرار.
- في حال عدم نجاح المقارنة يتم إستدعاء التابع مرة أخرى حتى تنجح هذه المقارنة.

قليلة جداً هي الامثلة التي تستخدم المتغيرات الساكنة في موضوع العودية لإنهاء الاستدعاء الذاتي للتابع ، هناك شروط أخرى أكثر تقنية وابتكاراً من مجرد تشبيه العودية بحلقة for ، سنتعرض لها في المثال التالي .
وبالرغم من أن حلقات التكرار أفضل بكثير من العودية والسبب في ذلك أن العودية تستهلك كثيراً من الطاقة فالأفضل هو أن تترك هذا الموضوع (أي موضوع العودية) لمهاراتك البرمجية وألا تستخدمه إلا في حالات استثنائية حينما لا تجد حلاً إلا بهذا الموضوع ، وهناك بالفعل بعض الأشياء التي لا يمكن حلها إلا بموضوع العودية.

مثال عملي:
هذا هو المثال الوحيد الذي سأتناوله عن موضوع العودية للأسباب التي ذكرتها سابقاً.
سنقوم بكتابة كود يحسب مضروب أي عدد ما ، وسنحله بطريقة التكرار وليس بطريقة العودية.
إليك أمثلة على مضروب أي عدد إن كنت لا تفهم ما هو:
2! = 2 * 1;
5! = 5 * 4 * 3 * 2 * 1 ;
أول ما يجب علينا التفكير فيه هو معرفة متى سيتوقف التابع عن استدعاء نفسه، كما تعلم أن مضروب الصفر يساوي الواحد الصحيح (0! = 1) .
بالتالي فحينما يصل التابع إلى الرقم 0 فسيتوقف عن استدعاء نفسه.
أما عن كيفية سيصل هذا التابع إلى الصفر فالجواب بسيط حينما يقوم بمقارنة العدد الممرر بالصفر وفي حال لم يجده كذلك فإنه يقوم بإنقاص العدد الممرر رقماً واحداً ثم يمرره إلى التابع المستدعى الآخر وهكذا:

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int fact(int );
5.
6. int main()
7. {
8.    int i=0;
9.    cout << "Enter the Number:\t";
10.         cin >> i;
11.
12.         int x=fact (i);
13.         cout << x << endl;
14.
15.         return 0;
16.   }
17.
```

```
18.   int fact (int x)
19.   {
20.         if (x==0) return 1;
21.         else return x*fact(x-1);
22.   }
```

- يطلب البرنامج من المستخدم إدخال العدد الذي يريد إيجاد مضروبه في السطر 10.
- يتم إنشاء المتغير x والذي سيتم تخزين نتيجة حل هذا المضروب فيه ، وسيتم تهيئته بالقيمة العائدة للتابع fact ، والذي ستتم تمرير العدد الذي أدخله المستخدم لحساب مضروبه.
- ينتقل التنفيذ إلى السطر 20 ، وفيها يقارن البرنامج العدد الممرر بالعدد 0 وفي حال كان كذلك يقوم بإعادة القيمة 1 ، لأن مضروب الصفر هو العدد الصحيح.
- في حال لم يكن كذلك فإن التابع يعيد قيمة ضرب العدد الممرر في مضروب العدد الذي قبله فلو كان العدد الممرر هو 5 فيمكن تشبيه قيمة الإعادة رياضياً هكذا (5 * 4!) أما عن كيف كتابتها برمجياً فهو بتمرير الرقم 4 إلى تابع من نفس التابع fact مرة أخرى وهكذا تكون العملية متتالية حتى يجد البرنامج الرقم 0 حينها سيعيد القيمة 1 وبالتالي ينتهي كل شيء.

في حال ما لم تفهم ما سبق فقم بإعادة قراءته من جديد لأنه مهم في بعض الامور والتي نادراً ما ستواجهها .
أما إذا فهمت ما سبق فسأترك لك هذا المثال الآخر والذي يقوم بطباعة السلسلة fibbianci .
ملاحظة: هذه السلسلة الحسابية عبارة يكون العدد عبارة عن مجموع العددين الذين قبلاه في السلسلة مع العلم أن العدد الاول والثاني هما 1، انظر:

1 1 2 3 5 8 13 21 34 55 .... etc

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int fib(int );
5.
6. int main()
7. {
8.
9.   int i=0;
10.        cout << "Enter the Number:\t";
11.        cin >> i;
12.
```

```
13.         i=fib (i);
14.         cout << i <<  endl;
15.
16.         return 0;
17.   }
18.
19.   int fib (int x)
20.   {
21.         if ( x<3)
22.               return 1;
23.         else return (fib (x-2) + fib (x-1) );
24.
25.   }
```

يقوم هذا البرنامج بطباعة رقم السلسلة الذي أدخلت موقعه منها.

التحميل الزائد للتوابع:

يعتبر هذا الموضوع هو أول نوع من أنواع تعدد الاوجه والتي هي ثلاثة أنواع وتعدد الأوجه أحد أساسيات البرمجة الكائنية ، وهذا يعني أنه لن يمكنك تطبيق هذا الموضوع على لغة السي (إن وجد مترجمات للغة السي مستقلة عن مترجمات السي بلس بلس).

كما قلت أن من أحد أهم أهداف البرمجة الكائنية هو الوصول إلى استقلالية الكود الذي تكتبه وإمكانية إعادة استخدامه وسهولة فعل ذلك ، والتحميل الزائد يعد أحد الأساليب القوية لفعل ذلك.

التحميل الزائد للتوابع يعني وجود نسخ أخرى تحمل نفس اسم التابع الزائد التحميل ولكنها تختلف إما في عدد الوسائط أو نوع الوسائط أو حتى ترتيب هذه الوسائط.

والفائدة من ذلك تظهر فيما لو فهمت موضوع الوسائط الافتراضية ، فوجود الوسائط الافتراضية في التوابع يمكنك من استدعاء الدالة بطريقتين مختلفتين إحداها بدون ذكر قيم الوسائط الافتراضية والأخرى بتغيير قيم الوسائط الافتراضية ، لنفرض أنك قررت كتابة أحد التوابع وهو التابع Find ، وتريد من هذا التابع أن يقوم بالبحث في أي مصفوفة يطلبها المستخدم مع العلم أن هناك مشكلة كبيرة وهي كيفية جعل هذا التابع يتعامل مع جميع أنواع المصفوفات int و char و float ... وغيرها الحل الوحيد هو أن تقوم بزيادة تحميل التابع find ، أي ستصبح النماذج المصغرة لنسخ التابع find، هكذا:

```
int find (int [] , int );
char find (char [] , char);
float find (float [] , float );
```

وحينما تصل لمرحلة تعريف هذه التوابع ، فيجب عليك تعريف كل نموذج على حدة ولن يكفيك تعريف تابع واحد فحسب.

عليك أن تعلم أن التحميل الزائد لأي تابع يعني أن هناك إصدارات أو نسخ أو توابع أخرى تحمل نفس اسم هذا التابع ولكنها تختلف في الوسائط سواء في العدد أو النوع.

سنقوم الآن بتقليد التابع Abs الذي يعيد القيمة المطلقة لأي عدد تدخله من المكتبة stdio في لغة C ، ولربما تقوم أنت بتطويره حتى يصبح أفضل من التابع الموجود في لغة C :

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int Abs (int );
5. float Abs(float );
6. double Abs (double );
7.
8. int main()
9. {
10.         int Int=0;
11.         float Float=0;
12.         double Double=0;
13.
14.         cout << "Int:\t" ; cin >> Int;
15.         cout << "Float:\t"; cin >> Float;
16.         cout << "Double:\t";cin >> Double;
17.         cout << endl << endl;
18.
19.         cout << "Int:\t" << Abs(Int) << endl;
20.         cout << "Float:\t" << Abs (Float) << endl;
21.         cout << "Double:\t" << Abs(Double) << endl;
22.         cout << endl;
23.
24.         return 0;
25.   }
26.   int Abs(int X)
27.   {
28.         return X<0 ? -X : X;
29.   }
30.
31.   float Abs(float X)
32.   {
```

```
33.         return X<0 ? -X : X;
34.    }
35.
36.    double Abs (double X)
37.    {
38.         return X<0 ? -X :X;
39.    }
```

- انظر إلى النماذج المصغرة للتوابع Abs( ) ، جميعها تأخذ أنواعاً مختلفة وسيقوم المترجم حينما تقوم باستدعاء هذه التوابع بالبحث عن التابع المناسب ، النماذج موجودة في الأسطر 3 و 5 و 6.
- في الأسطر 10 و 11 و 13 تم الإعلان عن ثلاث متغيرات من الأنواع int و float و double وتسمية كل متغير بنفس مسمى نوعه ولكن بجعل الحرف الأول كبيراً والسبب في هذا الإجراء حتى تستطيع التفريق بينها في البرنامج
- تطلب الأسطر 14 و 15 و 16 منك إدخال قيم هذه المتغيرات ، حتى تستطيع فيما بعد إيجاد القيمة المطلقة لكل عدد.
- السطر 19 يقوم بطباعة القيمة المطلقة للمتغير من النوع int ، وكما ترى فهو يقوم بطباعة القيمة العائدة للتابع int Abs( ) ، وكما ترى فإن التنفيذ سينتقل إلى البحث عن التابع المناسب لمثل هذا النوع من الوسائط والتابع الأفضل هو في السطر 26 .
- في السطر 28 ، يقوم البرنامج بمقارنة العدد الممرر (الذي نود إيجاد القيمة المطلقة له) مع الصفر وفي حال كان أصغر فإننا نعيد العدد ولكن بقيمة سالبة وبالتالي فعندما تدخل العدد 2- فإن المقارنة ستنجح وبالتالي سيقوم التابع بإرجاع القيمة بعد إضافة السالب إليها أي ستصبح القيمة العائدة هكذا 2 - - ، والتي رياضياً تساوي 2 ، أما في حال لم تنجح المقارنة أي أن العدد أكبر من الصفر أو مساوي له فسيعيد التابع نفس القيمة ويقوم التابع main( ) بطباعتها في السطر 19 .
- نفس الأمر سيحدث في السطرين 20 و 21 .

بالرغم من سهولة هذا الموضوع إلا أنه يعتبر أحد أهم الإمكانات في لغة السي بلس بلس وفي البرمجة الكائنية بشكل عام ، وخاصة حينما تبدأ في التعامل مع الكائنات.

محاذير عند التحميل الزائد للتوابع:

هناك بعض الأخطاء عندما تقوم بالتحميل الزائد للتوابع ، والتي يغفل عنها الكثيرون ، وهذه هي أهمها:

1- لن يكون بإمكانك زيادة تحميل أي تابع اعتماداً على القيمة العائدة فقط ، تعتبر هذه الإعلانات عن التوابع خاطئة:

```
int Abs(int , int );
float Abs( int , int );
```

والسبب بسيط وهو أن المترجم لن يعلم أبداً ما هو التابع الذي سيقوم باستدعاءه بالضبط ، لأن الوسائط هي نفسها.

2- لن يكون بإمكانك زيادة تحـميل أي تـابع فـي حـال كانـت لـه نفـس قائمـة الوسائط حتى وإن كانت بعض وسائطة افتراضية ، أنظر إلى هذا المثال:

```
int function(int a ,int b);

int function(int a,int b ,int c=100);
```

والسبب أنه حين استدعاء هذا التابع بواسطـة وسيطين ولـيس ثلاثـة فحينهـا لن يعرف المترجم أي تابع يستدعي.

3-أيضاً لن يكون بإمكانك زيادة تحـميل تابع على هذا الشكل:

```
int function(int a);

int function(const int a);
```

تذكر لكي ينجح التحـميل الزائد للتوابع ، فعلى التوابع التي تحمل نفس اسم التابع أن تختلف في قائمة الوسائط سواء في العدد أو الترتيب أو النوع أو أي شـيء آخر مع الأخذ بعين الاعتبار المحاذير السابقة.

التوابع السطرية Inline Function :

حينما تقوم بكتابة تابع ما ، وستقوم في الكــود باسـتدعاء هـذا التـابع خمـس مرات فسيقوم المترجم بجعل هذا التابع في مكان خاص له بالذاكرة ، ثم مـع كل استدعاء لهذا التابع ينتقل التنفيذ إلى تلك المنطقة من الذاكرة ، وبالتـالي فالذي يوجد في الذاكرة هـو نسخـة واحدة من التابع ، ولو قمت باسـتدعاءها مليون مرة.

يقلل هذا الإجراء من السرعة كثيراً بسبب هذه الاستدعاءات ، وخاصة إذا كان التابع عبارة عـن سطر أو سطرين فربما يكون من الأفضـل الـتخلص مـن هـذا الاستدعاء عبر التخلص من التابع وكتابة الأوامر التي نريـد كتابتهـا فـي التـابع الرئيسـي ولكن هذا الشيء غير مفضل ولا ينصح به لأننا سنفقد القدرة علـى الاستفادة من هذا التـابع مسـتقبلاً ، لـذلك فسـيكون مـن الأفضـل جعـل هـذا التابع تابعاً سطرياً وفي حال قمت بجعله سطرياً فإن المترجم سيقوم بنفس الإجراء السـابق الـذي كنـا نـود إضـافته لحـل المشـكلة أي نسـخ الأوامـر إلـى التابع الرئيسـي ، ولكنك ستتعامل معها على أنها تابع حقيقي.

الفائدة الحقيقية للتوابـع ليسـت علـى مسـتوى التصميم بـل علـى مسـتوى كفاءة البرنامج فالتابع المكون من سطرين سيكون من الأفضـل الـتخلص مـن استدعائها لأن ذلك سيؤثر على سرعة البرنامج وجعله ضمن التابع الرئيسي بجعله تابعاً سطرياً .

لا تقـم بجعـل جميـع توابعـك سـطرية، لأنـك حينمـا تقـم بـذلك سـيزيد حجـم الذاكرة بشكل كبير جداً وستزداد السرعـة (ولكـن لـن تسـتفيد مـن السرعـة بسبب زيادة حجم البرنام) التوابع التي قد تجعلهـا سـطرية هـي تلـك التوابـع الصغيرة التي لا تزيد عـن سطرين أو سطر.

الإعلان عـن تابع سطري يكون بكتابة الكلمة inline قبل نوع التابع انظـر إلـى هذا المثال:

```
inline int function( ) ;
```

تعريف قوالب التوابع:

سنبدأ بداية من البرمجة الهيكلية ؛أقصد هـنا مـن موضـوع التوابـع ، سـنقوم بالتخلص مـن عقـدة التحــميل الزائـد للـدوال عبـر دالـة واحـدة وعبـر موضـوع القوالـب ثـم سـنتقدم أكثـر إلـى موضـوع الأصـناف والكائنــات فـي الوحـدات القادمة.

لنفرض أنك تريد كتابة دالة تقوم بإيجاد القيمة المطلقة لرقم معين ، بالرغم من أن هذه الدالة موجودة في المكتبات القياسية للغة السي إلا أننا سنقوم بكتابتها من جديد ، فإنك لن تجد أفضل من المعامل الشرطي الثلاثي ليقوم بالمهمة على هذا النحو:

```
int Abs(int X)
{
      return X<0 ? -X : X;
}
```

وكما ترى فإن هذه الدالة لا تتعامل إلا مع الأعداد إلا من النوع int ، وقد تقوم بزيادة تحميل هذه الدالة حتى تتعامل مع بقية الأنواع (كما في الامثلة السابقة في هذه الوحدة) ؛ وقد تجد هذا العمل مملاً كما أنه يضيع المزيد من الوقت والجهد في أمور كان من الأفضل للحاسب أن يتعامل معها هو بنفسه دون أن يترك للمبرمج التعامل مع هذه التفاصيل الصغيرة وقد تجد الأمر متعباً للغاية حينما تتعامل مع دوال أخرى أكثر تعقيداً من حيث عدد الوسائط وأنواعها مما يلزمك بكتابة جميع تلك الإحتمالات.
توفر لك السي بلس بلس طريقة أفضل من ذلك بكثير ألا وهي القوالب ، دعنا الآن نقوم بقولبة الدالة السابقة حتى تصبح قادة على التعامل مع جميع الاحتمالات:

CODE

```
1- template <class T> T Abs(T X)
2- {
3-       return X<0 ? -X : X;
4- }
```

التغيير الحاصل هو في أول سطر من التابع حيث تغير من int Abs (int X) إلى:

```
.       template <class T> T Abs(T X)
```

قارن بين السطرين الاولين في التابعين ؛ تجد أنه لا وجود للنوع int بل الحرف T ؛ والحرف T في الحقيقة هو نوع الوسائط ونوع القيمة المعادة كما هو في تعريف التابع ؛ وليس الأمر في أن هناك نوع بيانات جديد هو T بل لأننا قمنا بقولبة الدالة ففي السطر الأول قمنا بكتابة الكلمة الأساسية وهي template ومعناها أننا نخبر المترجم بأن التابع القادم نريد قولبته ، ثم يأتي بعض ذلك وبين قوسين حادين الكلمة الأساسية <class T>
لاحظ أنه بإمكانك تغيير الحرف T إلى ما تريد لكن الكلمة الأساسية class لا تستطيع تغييرها إلى ما تريد وهذه الكلمة بمفهوم عام أنك تخبر المترجم أن الحرف T هو نوع بيانات على المترجم تحديده بنفسه ولا يجب ذلك على مبرمج أو مستخدم التابع وبالمعنى فإنك إذا قمت بتمرير إحدى القيم من النوع int فإن المترجم يقوم بإصدار دالة تستخدم النوع int ويستخدمها وهكذا بالنسبة لجميع الأنواع وحتى النوع char .

ملاحـظـة ضروريـة:
الآن يجب عليك التفريق بين معامل دالة عـادي ونــوع بيانـات خـاص بالدالـة ، المعامل العادي بالنسبة للدالة هـو عبارة عـن قيمـة معروف نـوع بياناتهـا أمـا نـوع البيانات الخاص بالدالة فهـذا يتم حلــه فـي وقـت الترجمــة حينما يقـوم البرنامج بإصدار نسخ مختلفـة من هذه الدالة حسب النـوع الممرر للدالة.

كيف يعـمل المترجم في حالة القوالب:

في التابع السابق فإن القالـب الســابق لا يقـوم بتوليـد إصـدار كــودي للتـابع Abs( ) ؛ لأنـه بكل بساطة لا يعرف ما هي أنـواع البيانات التي يتعامل معهـا ، ولكن حينما تقوم بإستدعاء هذه التابع عبر تابع آخر فإن الأمـور تتضح ويعـرف المترجم مـا هــو نــوع الوسـائط وبالتـالي يقـوم بإصـدار نسـخة تـابع جديـدة تستخـدم نفس ذلك النـوع من الوسائط وسيستخـدم هذا الإصـدار حتـى لـو قمت بإستدعائها مرة أخـرى أما في حال أنـك قمـت بإسـتدعائها مـرة أخــرى ولكن هذه المرة بنـوع وسائط مختلف فإنـه لا يقوم بإسـتخـدام التـابع المصـدر سابقاً بل ينتج تابع جديد كلياً ومستقل عـن التابع الاول.

ما هـو القالب:

كما رأيت فإنـه في الحقيقـة ليس هـناك تابع في الكـود السابق بـل مخـطـط توضيحي يقـوم بإخبـار المتـرجم كيـف يتـرجم هـذه الدالـة لـيس إلا ، والقـالـب نفســه لا يقوم بحجز ذاكرة بل حينما يقوم المترجم بترجمـة ذلك التـابع إذا مـا كـان هـناك إستدعاء أو أي شــيء آخـر ؛ وحتـى تتأكـد مـن كلامـي هـذا فقـم بكتابة أي شـيء ضمن تعريف قالب الدالـة لكـن لا تقــم بإسـتدعائها وستجـد المترجم لا يقول شـيء بخصوص ذلك.

الآن ما رأيك بأن نتقدم أكثر فـي هـذا المجـال ونقـوم بكتابـة قالـب تـابع يقـوم بالمقارنـة بين عـددين اثنين وتقوم بإرجاع الأكبر:

```
template <class T> T Big(T n1,T n2)
{
   return n1 >n2 ? n1 : n2;
}
```

يقوم هذا القالب بالمقارنـة بين عـددين اثنين وسينجح في مختلف الأحــوال ، لكن ماذا لو قـام المستخــدم بإدخـال عــددين مـن نـوعين مختلفـين بـدلاً مـن عـددين اثنين بنوع واحد مثلاً أدخل العــدد الاول 1 والعـدد الثـاني 0.5 ؛ وكمـا تـرى فـإن النــوع الأول هــو int والنــوع الثـاني هــو float ، الــذي سـيفعله المترجم حينما يقوم بترجمـة الإصدار الخاص بهذا التابع أن هـناك خــطـأ وهــو أنك قلت حسب تعريف التابع أنــه لـن يكـون هــناك سـوى نـوع بيانـات واحـد والآن فإن هـناك نـوعين اثنين من البيانات؛ وحتى تستطيع تعـديل هذا الخـطأ فكل ما عليك هـو إضافـة القليل ؛ أنظر الآن إلى هذا القالب:

CODE

```
1- template <class T1,class T2> T1 Big(T1 n1,T2 n2)
2- {
3-       return n1 >n2 ? n1 : n2;
4- }
```

في أول سطر أصبح هناك كلمتين أساسيتين من الكلمة class ؛ كل كلمة تدل على نمط بيانات قد يكون مختلف وقد يكون هو نفسه، فالمعامل T1 يدل على نوع المعامل الأول والمعامل T2 يدل على نوع المعامل الثاني ؛ وهكذا فإن المشكلة أصبحت محلولة.

ملاحظة ضرورية للغاية:
لو إفترضنا أنك قررت عدم إستخدام النمط T2 في تعريف الدالة فإنك حينما تقوم بإستدعاء الدالة فإن المترجم لن يدري ما هي بالضبط T2 هل هي نمط بيانات أم قيم أم شيء آخر وبالتالي فإنه يقوم بإصدار خطأ.

بإمكانك الآن إنشاء مكتبة يعتمد عليها ولو أنك ستقوم بها بشكل هيكلي على هيئة توابع إلا أنك تعتمد على بعض خواص البرمجة الكائنية مثل تعدد الأوجه والقوالب وغيرها وبإمكانك جعل هذه المكتبة مكتبة يعتمد عليها ما رأيك مثلاً بكتابة مكتبة تقوم بالبحث في أي مصفوفة تمرر إليها أو دالة تقوم بتحويل الأعداد إلى الأنظمة الأخرى وغير ذلك.

زيادة تحميل القوالب:
بالرغم من الفائدة العظمى للقوالب إلا أنك ستجد أنه من الغباء أن تقوم الدالة Abs( ) بمعالجة حروف وليس أعداد من أجل ذلك فبإمكانك أن تقوم بكتابة دالات أخرى من دون أي قولبة تستقبل وسائط من النوع char حتى تقوم بإنهاء البرنامج وليس بجعله هكذا يعالج جميع أنواع المتغيرات.

اصنع مكتباتك الخاصة (ملفات البرمجة):
ملف البرمجة هو ملف يحتوي على أوامر وستتم ترجمته بصورة منفصلة.
لطالماً استخدمنا ملفات البرمجة وأكبر مثال على ذلك أننا دائماً نقوم بكتابة السطر التالي:

```
#include <iostream>
```

الفائدة من استخدام المكتبات أو الملفات المنفصلة:
هناك بضعة فوائد كبيرة:
- أن التوابع التي تقوم بكتابتها لن تضيع وستقوم باستخدامها مرات ومرات كثيرة ، انظر إلى التابع Abs( ) .
- عندما يصبح البرنامج الذي تقوم بكتابته كبيراً فلن تضطر عند التعديل إلا إلى إعادة ترجمة الملف الذي تم التعديل فيه وليس كل البرنامج.

ملف الرأس Header File:
يحتوي ملف الرأس فقط على الإعلانات وليس التصريحات أو التعريفات ، أي النماذج المصغرة للتوابع فقط.
قم الآن بتشغيل برنامج الفيجوال سي بلس بلس ثم اذهب إلى الخيار File، وانقر على New ثم عبر علامة التبويب Files ثم باختيار C/C++ header file وانقر على OK . سنقوم الآن بإنشاء ملف رأس نقوم فيه بكتابة التابع Abs ، انظر إلى هذا الكود .

CODE

```
1- #ifndef AbsModule
2- #define AbsModule
```

```
3- int Abs(int );
4- float Abs( float );
5- double Abs (double );


6- #endif
```

في السطر الأول يحتوي على توجيه للمعالج (مرحلة ما قبل الترجمة) وهو يخبر المعالج ، إذا لم يقم أي ملف برمجة آخر بتعريف الاسم التالي AbsModule ، فقم بالسماح للمترجم بترجمة الأسطر حتى يجد الكلمة endif الموجودة في السطر السادس وحينها يتوقف.
قد تتساءل عن سبب هذا الإجراء والسبب في ذلك حتى نمنع أي كان من آثار تضمين هذا الملف عدة مرات ، فلو افترضنا أننا نعمل على برنامج ضخم وفي أحد ملفات البرمجة قمنا بكتابة هذا الأمر:

```
#include "AbsModule.h "
```

ثم ولأن أحد المبرمجين الآخرين نسي فقام في ملف برمجة أخر بكتابة هذا السطر :

```
#include "AbsModule.h "
```

فحينها سيكون لدينا ست نماذج مصغرة للتابع Abs ، وبسبب أن هنا نموذجين مصغرين متشابهين فسيقوم المترجم بإصدار خطأ ، ولربما لن تكتشف أنت هذا الخطأ بتاتاً.

السطر 2 ، يتابع السطر الأول فهو يقول للمعالج قم بتعريف هذا الاسم AbsModule . أي إذا جمعنا السطرين الأول والثاني فإن السطر الاول يقول إذا لم يكن هناك أي تعريف للمسمى AbsModule فتابع ترجمة هذا الملف وبالتالي فإن السطر الثاني سيقول قم بتعريف AbsModule .

أتمنى أن تكون فهمت هذه النقاط المهمة للغاية ، الآن انظر إلى الأسطر 3 و 4 و 5 تعتبر هذه الأسطر هي أهم ما في ملف البرمجة وهي فقط تحتوي على إعلانات ليس إلا ، لا تقم بجعلها تحتوي على تعريفات.

ملاحظة بالنسبة للأسطر 1 و 2 و 6 فقم بإلغاءها حالياً ولربما نقوم بإعادة ذكر هذه المواضيع حينما نصل إلى موضوع الأصناف ، ولكن احرص على فهم ما تعنيه وهذه الأسطر يطلق عليها مسمى حراس التضمين، ربما لن يتضمن هذا الكتاب شرحاً متكاملاً للمكتبات التي تقوم بإنشاءها وكيف تتعامل مع مساحة الأسماء وحراس التضمين وما إلى ذلك.

إنشاء ملف التنفيذ:

ملف التنفيذ هي الذي يحتوي على تعريفات ما يحتويه ملف الرأس ، هذا هو ملف التنفيذ لملف الرأس AbsModule :

CODE

```
1. #include <iostream>
2. #include "AbsModule.h"
```

```
3. using namespace std;
4.
5.
6. int Abs( int X)
7. {
8.    return X<0 ? -X : X;
9. }
10.
11.   float Abs( float X)
12.   {
13.         return X<0 ? -X : X;
14.   }
15.
16.   double Abs ( double X)
17.   {
18.         return X<0 ? -X :X;
19.   }
```

في السطر الاول قمنا بتضمين المكتبة iostream ، حتى نستطيع استخدام مساحة الأسماء std في السطر 3 .
في السطر الثاني قمنا بتضمين ملف الرأس AbsModule ، وهناك نقطة مهمة للغاية عليك تذكرها دائماً ، انظر إلى كيفية تضمين ملف الرأس الذي أنشأناه ، لقد قمنا بوضع الاسم بين علامتي تنصيص (" ") وليس بين قوسين حادين كما في المكتبات القياسية والسبب في هذه الطريقة حتى يعلم المترجم أن هذه المكتبة في نفس المجلد الذي فيه الكود لأن البرنامج لن يقوم بالبحث عنها في جميع نظام التشغيل ، أيضاً لاحظ أننا قمنا بوضع علامة الامتداد (.h) .
من الأسطر 6 إلى 19 احتوى على تعريفات النماذج المصغرة للتوابع في ملف الرأس AbsModule .

الآن بقي علينا كتابة ملف البرمجة main.cpp ، والذي سنختبر فيه صحة هذه المكتبة ، انظر إلى هذا الكود:

CODE

```
1. #include <iostream>
2. #include "AbsModule.h"
3. using namespace std;
4.
5. int main()
6. {
7.    int Int=0;
8.    float Float=0;
```

```
9.    double Double=0;
10.
11.         cout << "Int:\t" ; cin >> Int;
12.         cout << "Float:\t"; cin >> Float;
13.         cout << "Double:\t";cin >> Double;
14.         cout << endl << endl;
15.
16.         cout << "Int:\t" << Abs(Int) << endl;
17.         cout << "Float:\t" << Abs (Float) << endl;
18.         cout << "Double:\t" << Abs(Double) << endl;
19.         cout << endl;
20.
21.         return 0;
22.   }
```

لن أشرح ما يحويه هذا الكود فقد شرحته سابقاً في مثال كودي آخر من هذه الوحدة ، كل المهم هو أننا قمنا بتضمين المكتبة التي أنشأنها في السطر 2 .

مؤشرات التوابع:

من الممكن أن نعرف مؤشراً إلى تابع ثم يصبح بإمكاننا التعامل مع هذا المؤشر كأي مؤشر آخر. يمكن أن نسند له قيمة أو نخزنه في مصفوفة أو نمرره كوسيط ...إلخ.

سيمكننا هذا من إنشاء مصفوفة متكاملة من التوابع وليس من المتغيرات.

لاحظ هنا أننا لا نتحدث بالتحديد عن موضوع التوابع بل كل الذي نتحدث عنه هو أنه بإمكانك إنشاء مؤشر يشير إلى أحد التوابع ، هذه الميزة تمنحك الكثير من الاختصار في الكود ومن الجهد ومن الوقت ، بإمكانك الإعلان عن مؤشر إلى تابع هكذا:

```
int (*function) (int , int)
```

لاحظ أننا نتحدث هنا عن مؤشر وبالتالي فالذي تراه ليس تابعاً أو نموذج مصغر لتابع بل هو مؤشر بإمكانه الإشارة إلى أحد التوابع التي تعيد نفس القيمة وتستقبل نفس البارامترات كما في التصريح عن المؤشر.

سنقوم الآن بكتابة برنامج شبيه بالآلة الحاسبة يقوم بالعمليات الحسابية الأساسية وكل عملية سنقوم بوضعها في تابع وسترى كم من الأسطر اختصرنا لو أننا لم نستخدم مؤشرات التوابع.

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. double plus(double , double );
5. double del(double , double );
6. double multipy(double ,double );
```

```
7. double divide(double ,double);
8.
9. int main()
10.   {
11.         double Num1,Num2,Value;
12.         char Operator;
13.         double (*Function) (double ,double );
14.
15.         cout << "Please Enter Num1:  ";cin>>Num1;
16.         cout << "Please Enter Operator  ";cin>>Operator;
17.         cout << "Please Enter Num2:  ";cin >>Num2;
18.
19.         switch (Operator) {
20.         case '+': Function=plus;break;
21.         case '-': Function=del;break;
22.         case '*': Function=multipy;break;
23.         case '/': Function=divide;break;
24.         default: cout << "\nBad Command\n";return 0;
25.         }
26.         Value = Function (Num1,Num2);
27.
28.         cout << "Tne Value is:  " << Value << endl;
29.
30.         return 0;
31.   }
32.
33.   double plus (double a,double b)
34.   {
35.         return a+b;
36.   }
37.   double del(double a, double b)
38.   {
39.         return a-b;
40.   }
41.   double multipy(double a, double b)
42.   {
43.         return a*b;
44.   }
```

```
45.    double divide(double a,double b)
46.    {
47.          return a/b;
48.    }
```

بالرغم من كبر حجم هذا الكود إلا أنه اختصر أكثر من 15 سطراً لو لم نستخدم مؤشرات التوابع.
في السطر 13 قمنا بالإعلان عن مؤشر إلى تابع ولم نحجز له أي ذاكرة.
تقوم الحلقة switch في السطر 20 بإختبار المتغير Operator وحسب الحرف المدخل أو العملية الحسابية المدخلة يتم إسناد المؤشر إلى تابع Function ، إلى أحد التوابع الأربعة في الأسطر من 4-7 .
انظر إلى كيفية عملية الإسناد في الجملة switch ، تجد أنها شبيها بعملية إسناد المتغيرات.

ملاحظة مهمة للغاية:
إذا ما قمت بإنشاء مؤشر إلى تابع فعليك بوضع قوسين بين اسم هذا المؤشر هكذا:

```
int (*Function) (int ,int);
```

أما إذا أردت الكتابة هكذا:

```
int* Function (int , int) ;
```

فسيظن المترجم أنك تقوم بالإعلان عن تابع يعيد مؤشر من النوع int .

صفوف التخزين Storage Classes :

مفهوم أو مصطلح صفوف التخزين يناقش في السي بلس بلس العلاقة بين المتغيرات والتوابع.
صفوف التخزين بالنسبة للمتغيرات تناقش عن التوابع التي ستسمح لهذه المتغيرات بالتفاعل أو الدخول ضمن تابع ما ، وكم ستبقى هذه المتغيرات حتى تنتهي دورة حياتها .
هناك ثلاثة أنواع من المتغيرات هي:
1- المتغيرات الآلية Automatic Variables :
2- المتغيرات الخارجية External Variables :
3- المتغيرات الساكنة Static Variables :

المتغيرات الآلية Automatic Variables :

أي متغيرات تعرف ضمن تابع ما تعتبر متغيرات آلية سواء أكان التابع main أو غيره .
بإمكانك القول صراحة ضمن الإعلان عن المتغيرات أنها متغيرات آلية ، انظر إلى هذا السطر:

```
auto int Variables ;
```

ولن يشتكي المترجم أو يقوم بإصدار أي أمر ما.
ولكن المترجم يقوم بتعريف جميع المتغيرات على أنها متغيرات آلية ولن تحتاج لكتابة الكلمة auto .

عـمر المتغيرات الآلية : Lifetime of automatic Variables

عـمر هذه المتغيرات الآلية هـو بعـمر التابع التي تنتمـي إليـه ، فمـثلاً لـو كـان المتغير A ضمن التابع Functio فإنــه لـن يكـون هــناك أي قيمــة أو أي ذاكـرة للمتغير A حتى يـتم إسـتدعاء التـابع Functio وحينما ينتهـي هـذا التـابع مـن مهـمته فإن قيم المتغير A تنتهي أو تضيع.

هذا الوقت بين ولادة المتغير A وإنتهاء التابع يدعـي العـمر Lifetime .

الرؤيـة : Visibility

مصطلح الرؤيـة يعبـر عــن مجـال الرؤيـة لهـذا المتغيـر وقـد تـم شـرح قواعــد مجالات الرؤية لجميع المتغيرات في وقت سابق من هذه الوحدة.

المهـم في هذه الفقرة هـو معرفة الفارق بين المصطلحين ، مصـطلح الرؤيـة ومصطلح العـمر بالنسبة للمتغيرات.

بالنسـبة للـنوعين الآخـرين فقـد تـم شـرح خواصهــما فـي وقـت سـابق مـن الكتاب.

هـناك أيضـاً بعـض الملاحظـات الأخـرى ، بالنسـبة لنـوع المتغيـرات الخارجيـة فليسـت الرؤية فيها ضمن البرنامج كله بل ضمن الملـف أو ملـف الـرأس الـذي عرفت فيه فقط ، وإذا كنت تتعامل مع ملفات كثيرة فيجب عليك إعـادة تعريـف هذه المتغيرات في كل ملف تريد أن تكون مرئيـة فيـه وإعـادة تعريفها غريبـة بعض الشـيء انظر إلى هذا السطر:

```
extern int Num;
```

هذا السـطر يعـني أن المتغير num متغير من النـوع int إلا أن الكلمـة extern تعـني أنه لن يتم حجز ذاكرة له والسطر الحالي يعـني تصريحاً فقـط للمتغيـر Num ، والذي يعـني أن المتغير Num قد تم الإعلان والتصريح عـنه فـي ملـف آخر وأن التصـريح عــنه فـي هـذا الملـف يعــني إمكانيـة رؤيتــه ضـمن مجـال الملف.

تـذكر أيضـاً أن المتغيـرات الآليـة تخــزن فـي stack أمـا المتغيـرات السـاكنـة والخارجية أو العامـة فتخزن ضمن heap .

نوع المتغيرات	الآلية	الساكنة	الخارجية
نطاق الرؤية	التابع	التابع	ملف الرأس
العـمر	ضمن التابع	ضمن البرنامج	ضمن البرنامج
مكان التخزين	Stack	heap	heap

أساسيات التوابع:

- لكل تابع قيمة معادة وهذه القيمة المعادة تكون من أحد أنواع البيانات فقد تكون int **أو** flaot **أو** double .... **إلخ ؛ أما إذا كانت من النوع** void.

- التوابع التي من النوع void **فائدتها البرمجية تكاد تكون أفضل من فائدة التوابع الأخرى ، فهي تستطيع تغيير المتغيرات وإعادة أكثر من قيمة وليس قيمة واحدة وقد يستفاد منها في تخصيصها لطباعة بعض الجمل على الشاشة.**

- أغلب التوابع لها وسائط أو بارامترات وهي المتغيرات التي تدخل في التابع لكي تقوم بمعالجتها ، إذا كانت هذه المتغيرات عبارة عن متغيرات عادية فلن يحدث لهذه المتغيرات أي شيء يذكر وستفتصر فائدة التابع على القيمة المعادة ، لكن إذا قمت بتمرير مؤشرات أو إشارات أو أي شيء فأي شيء يقوم به التابع على هذه المتغيرات سيغيرها طوال حياة البرنامج.

قواعد مجالات الرؤية:

- حينما يتم التصريح عن أي متغير ضمن تعريف أي تابع فإنه يصبح متغيراً خاصاً بالتابع وبالتالي فلن تستطيع التوابع الأخرى الوصول إلى المتغير لأنه ليس من ضمن مجالات ريتها.

- إذا سبقت المتغير الخاص الكلمة static **فحينها سيصبح هذا المتغير متغيراً خاصاً ساكناً أي أن قيمة هذا المتغير ستبقى كما هي دون أي تغيير حتى حينما يتم الإنتهاء من تنفيذ التابع إلا أنه ما زال يخضع لنفس قواعد مجالات الرؤية بالنسبة للمتغير الخاص ، وبالتالي فحينما يتم إستدعاء التابع مرة أخرى فلن تصبح قيمة المتغير الساكن إلى** 0 **أو إلى** NULL.

- إذا تم الإعلان عن أي متغير خارج أي كتلة فحينها سيكون تابعاً عاماً وبإمكان جميع التوابع الاخرى التعامل معه وكأنه متغير خاص بها إلا أن ذلك لا يعني أن التوابع لا تستطيع تغيير قيمته بل تستطيع فعل ذلك ، فأي تغيير يقوم به أي تابع على هذا المتغير سيبقى حتى حينما ينتهي تنفيذ التابع.

مقدمة في البرمجة الكائنية المنحى

Introduction to Object Oriented Programming

بداية:

منذ أن بزغ فجر الكمبيوتر وابتدأ المبرمجون عملهم في برمجة البرامج ؛ كان عليهم أن يتعاملوا مع الكمبيوتر بواسطة اللغة التي يفهمها هو وهي الأصفار والآحاد ؛ وكانوا بالفعل يعملون برامجهم بواسطة الصفر والواحد ... مما جعل الأمر مقتصراً على النوابغ في البرمجة فلم يكن بمقدور أي شخص التعامل مع هذه البرمجة الصعبة .. ثم بعد ذلك بدأ عهد عصر لغات البرمجة والتي يوجد منها نوعان لغات البرمجة المنخفضة المستوى ولغات البرمجة العالية المستوى وقد امتازت الثانية بسهولتها مما جعلها تطور من مستوى البرمجة إلى أبعد حد ؛ وربما أن القفزة النوعية التي أحدثها هذا النوع من اللغات –حسب رأيي- هو أنه أصبح بإمكان المبرمج التركيز أكثر على حل المشكلة التي يواجهها بدلاً من الإهتمام بالأرقام والحروف التي يكتبها.

البرمجة الإجرائية:

وهذه أول تقنية ظهرت والتي كان من المفترض لها أن تظهر ؛ تركز البرمجة الإجرائية على إجراء البرنامج في خطوات واضحة ومحددة لا تحيد عنها وهي عبارة عن كتلة واحدة.. وبالرغم من أنها قدمت للمبرمجين الكثير إلا أن الأكواد تصبح أكثر تعقيداً حينما يتعامل الشخص مع مشاريع كبيرة الحجم. أيضاً لم يكن بإمكان الأشخاص العمل كفريق عمل لأنه لا يوجد تقسيم واضح في الكود .. فالكود عبارة عن كتلة واحدة.. أيضاً عند القيام بصيانة البرنامج فإن الأمر يصبح أكثر تعقيداً وخاصة عند تتبع سير البرنامج لمعرفة أين يوجد الخطأ مما جعل المبرمجين يشبهون هذا النوع من البرامج بأنه **معكرونة الأسباجيتي**.

البرمجة الهيكلية:

أتت البرمجة الهيكلية لحل المشاكل التي تعاني منها البرمجة الإجرائية إلا أنها لم تقدم الكثير؛ ولا أعتقد أنها قفزة نوعية في مجال البرمجة ، فهي لا تقوم بأي شيء سوى بتقسيم الكود إلى عدة أكواد أو إجراءات بالمعنى الأصح.... أيضاً لا يمكنك في بعض الحالات أن تعيد إستخدام هذه الإجراءات في برامج أخرى وفي بعض اللغات التي لا تملك ميزة المرجعيات والمؤشرات لا يمكن للإجراء ألا يعيد سوى قيمة واحدة.. أيضاً لا يمكنك إستخدام المتغيرات العامة بكثرة فهي تعقد البرنامج أكثر وتجعل من عملية تتبع سير البرنامج عملية مستحيلة.. وبسبب أن بعض الإجراءات تعتمد على وجود هذه المتغيرات العامة في البرنامج فلن يمكنك إعادة إستخدام هذا الإجراء في برامج أخرى لأن هذا الإجراء ليس مستقلاً كما يخيل للبعض... من أجل كل هذه العيوب والنواقص في البرمجة الهيكلية

والإجرائية ظهرت البرمجة الشيئية والتي لم ترتكز إلا على تغيير مفهوم البرمجة

البرمجة الشيئية:

ترتكز البرمجة الشيئية في وجودها ليس على إجراءات وإنما على وجود الأصناف والكائنات ؛ فالأصناف هي الوحدة الأساسية لأي برنامج يكتب بالبرمجة الشيئية ؛ تتألف الأصناف من متغيرات ودوال. وبمعنى برمجي شيئي بحت (دون التدخل في لغات البرمجة) فإن الصنف يتكون من شيئين اثنين هما : الخواص Attributes **و السلوك** Behaviors .

تعريف الصنف: هو عبارة عن قالب يعرّف مجموعة من الخواص والسلوك كما هي موجودة في العالم الحقيقي.

مثال برنامج تسجيل الطلاب في الجامعة:

فمثلاً لو أردنا القيام بعمل برنامج لتسجيل الطلاب في الجامعة فمن الممكن أن نقسم البرنامج إلى عدة أصناف وهي صنف الطالب ؛ صنف الكلية أو القسم ؛ صنف مسجل الطلاب (صنف عمادة القبول والتسجيل) ؛ وسنأخذ مثال صنف الطالب أولاً ، يتألف صنف الطالب من متغيرات ودوال، من أمثلة المتغيرات لدى الطالب درجة الطالب ، تقدير الطالب، عمر الطالب ومن أمثلة الدوال لدى الطالب ، دالة إختيار الكلية المرغوب بها أما بالنسبة لصنف الكلية أو القسم فمن أهم المتغيرات لديه هي اسم الكلية واسم التخصص والدرجة التي يقبل على أساسها الطالب أما بالنسبة للدوال فمن أهمها دالة القبول المبدئي (والتي تتأكد من توافق شروط القبول مع الطالب) ودالة القبول النهائي (وهي الدالة التي تفاضل بين الطلاب حسب معايير الكلية وبالتالي تقبل الطالب)؛ وبإمكاننا هنا وضع متغير جديد ألا وهو مصفوفة الطلاب المقبولين قبولاً مبدئياً ومتغير آخر هو مصفوفة الطلاب المقبولين قبولاً نهائياً ولن نتعرض هنا على دوال أخرى مثل دالة الفصل النهائي لأننا نكتب هنا برنامج لتسجيل الطلاب بالنسبة للصنف الأخير وهو صنف مسجل الطلاب أو بالمعنى الأصح عمادة القبول والتسجيل ؛ صنف عمادة القبول والتسجيل يتألف من هذه المتغيرات: مصفوفة الطلاب الراغبين بدخول الجامعة وتحوي أيضاً قائمة بالتخصصات المرغوبة ؛ ومصفوفة أيضاً تحوي أسماء الكليات وأقسامها ومن الدوال دالة تقوم بتسجيل الطلاب في قوائم القبول المبدئي(تتأكد من توافق الشروط العامة للجامعة مع الطلاب) ودالة أخرى تقوم بإرسال اسم الطالب ودرجة الطالب إلى الكلية ودالة ثالثة تستقبل أسماء الطلاب المقبولين قبولاً نهائياً في الكليات وبالتالي فبإمكاننا تمثيل هذه الأصناف في الأشكال التالية:



الكلية
المتغيرات الأعضاء:
اسم الكلية
درجة القبول المبدئي في الكلية
مصفوفة التخصصات في الكلية
مصفوفة الطلاب المقبولين مبدئياً
مصفوفة الطلاب المقبولين نهائياً
الدوال الأعضاء:
دالة القبول المبدئي
دالة القبول النهائي

عمادة القبول و التسجيل
المتغيرات الأعضاء:
مصفوفة الطلاب الراغبين بالدخول في الجامعة
مصفوفة التخصصات المرغوبة لكل طالب
مصفوفة الكليات و التخصصات
مصفوفة الطلاب المقبولين قبولاً نهائياً
الدوال الأعضاء:
دالة القبول المبدئي في الجامعة
دالة الإرسال
دالة تستقبل أسماء الطلاب المقبولين قبولاً نهائياً

كما تلاحظ فلقد أتممنا تصميم برنامج تسجيل الطلاب في دقائق قليلة ولم نحتاج فقط إلا للقليل من التركيز وقمنا بتمثيل واقعي للكائنات في البرنامج ؛ تخيل الآن مالذي سيحدث لو أننا قمنا بتصميم برنامج هيكلي تخيل مدى التعقيد الواقع في البرنامج وكيفية تتبع سير البرنامج وماذا علينا أن نضعه متغيرات عامة ومالذي لا نضعه متغيرات عامة وماهي المتغيرات التي نرسلها لكل دالة وماهي القيم المعادة .. إلخ
هذه هي فكرة الكائنات وهي فكرة تبسط من المسائل المعقدة لجعلها تبدو بسيطة وتجعل صيانة البرنامج أكثر تقدماً وسرعة.

إنشاء المثائل(إنشاء كائن) Creating a Class Instance:

الآن سنأتي إلى بعض النقاط المهمة.. كما تلاحظ فلقد كتبنا الصنف الطالب لكن لم نحدد في هذا التصنيف ما هو اسم الطالب لنفرض أن عدد الطلاب الذين اتوا للتقديم هم خمسمائة طالب فمالذي علينا فعله كل الذي عليه أن نعرفه قبل أن نعمل أي شيء أن نفهم الفرق بين الصنف والكائن .. الصنف مثل المخطط بينما الكائن هو تطبيق هذا المخطط . أي أنك لو قمت بكتابة صنف الطالب في برنامج ما ؛ فلن يحجز له المترجم أي ذاكرة بالرغم من وجود المتغيرات لأنك في الأساس تخبر المترجم أن هناك صنف جديد فقط. لا تخبره بأن يحجز لك مكان في الذاكرة بالتالي فكل ما عليك فعله هو أن تقوم بإنشاء كائن Object . الآن انظر لهذا السطر الكودي الخارج عن الموضوع وحاول أن تفهم ما أحاول أن أقول:

CODE

```
int x=5;
```

كما ترى فأنت تحجز للمتغير x ذاكرة من نمط int العلاقة بين المتغير ونمط البيانات هي نفس العلاقة بين الكائن والصنف ؛ بالإمكان اعتبار الصنف الطالب هو نفسه النمط int فيما بالإمكان اعتبار الكائن ولنسمه مثلاً محمد هو نفسه المتغير x . وهذا هو الفرق بين الصنف والكائن. وبالتالي لحل المشكلة السابقة فبالإمكان إنشاء مصفوفة كائنات نطلق عليها الطلاب المتقدمين وتحوي خمسمائة عنصر من الصنف الطالب.
الآن وبعد أن فهمنا ما هو الفرق بين الصنف والكائن وبعد أن شرحنا تعريف الكائنات بإمكاننا أن ندخل في مبادىء البرمجة الكائنية.

** مباديء البرمجـة الكائنيـة:

ترتكز البرمجـة الكائنيـة على ثلاث مبادئ:

1- التجريد :Abstraction

2- الوراثـة : Inheritance

3- تعدد الأوجـه أو الأشكال :

الآن بعـد أن ذكرن المبادئ الثلاث فسنقوم بشرحها شرحاً تفصيلياً في بقية مواضيع الكتـاب ، بالنسبة للمبدأ الاول وهـو التجريد فهـو أمر سأقوم بشرحـه طوال هذه الوحدة والوحدات القادمـة وسأركز الآن على موضوع الكبسلة Encapsulation **، ثم سنتعـمق هذه الوحدة أكثر في هذا المبدأ ، بالنسبة للمبدأ الثاني والثالث فسيتعرض الكتاب بنسب متفاوتـة لهـما خلال الوحدة ما بعـد القادمـة**

الكبسـلـة أو التغليف :Encapsulation

قبل أن نذكر فائدة الكبسـلـة فعلينا أن نحاول إيصـال مفهـومها إلـى القـارئ ؛ تعـرف الكبسلــة علـى أنها إخفـاء المعلومـات عــن المستخــدم أقصـد هــنا مستخدم الصنف (دعـك عـن لماذا الآن؟) من الممكن أن نشبه الصـنف علـى أنـه صندوق أسـود هذا الصندوق له معلومـات لاستخدامــه فـإذا أخـذنا مثـال الصراف الآلي فأنـت تقـوم بإدخـال بطاقـتـك البنكيـة ورقمهـا السـري لتجـري بعض العـمليات والتي لا يهـمك أن تعرفهـا وتخـرج لـك مـا تريـد مـن الصـراف ؛ بهذه الطريقة يمكن تشبيه الكبسـلـة أو التغليف ؛ لا يهـمك أنـت أن تعـرف مـاذا يحدث في الصراف وهـذا أحـد الأسـباب وهــناك سـبب آخـر وهــو أن البنـك لا يريدك أن تعبث بالصراف فـإذا كــان بإمكــانك تغيير برنـامج الصـراف وبالتـالي تغيير برنامج البنك على ما تشتهيه نفسك فقـد تحصـل كارثـة اقتصـادية فـي البلاد .. وهذا أيضاً على صعيد البرمجـة الكائنيـة فمن جهـة لا يهـمك ما يحدث داخـل الصـنف ومـن جـهـة أخـرى فإنـه لا ينبغــي لـك أن تعبـث بالمحتــويات الداخلية للصنف... وهذه هي فائدة الكبسـلـة..وعلى الصعيد الكـودي فهــناك كلمتــان public **والتـي تعــني أن الأعضـاء الـذين تحتهـا هــم أعضـاء عـامــة بالإمكـان التغيير فيهـم والكلمـة الأخــرى هـي** private **وتعــني أن الأعضـاء الذين تحتها هــم أعضـاء غيـر مـرئيين خـارج الطبقــة أي أعضـاء مكبسـلين أو مغلفين.**

لوقت طـويل كان مبرمجي البرمجـة الإجرائيـة (كالسـي مثلاً) يحـاولون تجميع الأوامر التي يقومون بكتابتها ضـمن قالـب واحــد. فمثلاً فـي برنـامج تسـجيل الطالب ، كان عليهـم وضع الثلاث الأصناف السابقـة ضـمن برنـامج واحــد دون أن يكون هـناك فارق بينهـم وليس عليه ذلك فحسـب ، بل عليه أيضاً محـاولة تنسيق عـمل الثلاث الكائنـات دون أن يكون له نظرة محـددة عـن هذه الأشياء الثلاث ؛ من أجل ذلك أتـت البرمجــة الكائنيــة والتـي جعلـت الـثلاث الكائنــات مفصولة عـن بعضها البعض مما زاد من تنظيم الكـود وطبيعتـه.

الآن وبعـد هذه المقدمـة حول البرمجـة الكائنية سندخل في الأكـواد وسنقوم بكتابة صنف الطالب في البرنامج:

CODE

```
1      class Student //   class كما تلاحظ فللإعلان عن الصنف نكتب كلمة
```

```
2	{
3	public: // لإعلام المترجم بأن الأعضاء الدارجة تحت هذا الاسم  أعضاء عامة
4	choosingCollege( );
5	SignUp( );
6	private: // لإعلام المترجم بأن الأعضاء الدارجة تحت هذا الاسم أعضاء خاصة
7	int itsGrade;
8	int itsAge;
9	char specialization[ ];
10	} ;
```

لن نكتب ما تحويه الدوال حالياً لأن المهم هو شرح الكود السابق كما ترى في السطر الأول بدأ الإعلان عن الصنف بكلمة class **ثم اسم الصنف وهو** student **بعد ذلك نبدأ في السطر الثاني بكتابة قوس الفتح وفي السطر العاشر نقوم بإغلاق هذا القوس أما في السطر الثالث فلقد أعلمنا المترجم أن الأعضاء في السطر الرابع والخامس هي دوال عامة بإمكان الطبقات الأخرى رؤيتها أما بالنسبة للأعضاء في السطر السابع والثامن والتاسع فهي أعضاء خاصة قمنا بكبسلتها لأننا لا نريد من أحد العبث بها وفي السطر الأخير قمنا بكتابة الفاصلة المنقوطة والتي لا تنسى كتابتها دائماً.**
لقد أعطاك المثال السابق فكرة عامة عن الإعلان عن الأصناف وكبسلة الأعضاء داخلها. الآن وبعد أن انتهينا من المثال السابق. فسندخل في أمثلة كودية أكثر جدية. سنقوم بإنشاء صنف كامل ونقوم بتنفيذه حتى تفهم تركيب الأصناف بشكل عام والصنف الذي سنقوم بصنعه هو عبارة عن آلة حاسبة بسيطة لن نسعى من خلالها إلى بناء مشروع آلة حاسبة كاملة بل مجرد إيصال الفكرة لديك فقط.

CODE

```
1	#include <iostream.h>

2	class maths
3	{
4	private:
5	float itsNum1;
6	float itsNum2;
7
8	public:
9	GetNum1Num2(float i,float j);
10	print();
11	};
12	maths::GetNum1Num2(float i,float j)
13	{
14	itsNum1=i;
```

```
15    itsNum2=j;
16    }
17    maths::print()
18    {
19     cout << "add:\n" << itsNum1+itsNum2 << endl;
20     cout << "subtract:\n" << itsNum1-itsNum2 << endl;
21     cout << "multiby:\n" << itsNum1*itsNum2 << endl;
22     cout << "divide:\n" << itsNum1/itsNum2 << endl;
23    }
24
25    int main ( )
26    {
27    float i,j;
28    cin >> i>>j;
29    maths a;
30    a.GetNum1Num2(i,j);
31    a.print();
32    return 0;
33    }
```

كما تلاحظ فإننا قمنا بإنشاء صنف أسميناه maths إن لهذا الصنف وظيفة محددة وهي حساب عددين بالعمليات الأربع الأساسية وطباعة جميع النتائج.
وكما ترى فهناك متغيران فقط في الصنف هما في السطر الخامس والسادس وهما بالطبع متغيرات خاصة أما بالنسبة للأعضاء العامة فهناك إجراءان فقط ؛ الأول يستخدم للوصول إلى المتغيرات الخاصة المكبسلة في الصنف والآخر يقوم بالحساب وطباعة النتائج.
في السطر الحادي عشر انتهى الإعلان عن الصنف وابتدأنا الآن بكتابة ما تحويه تلك الإجراءات وحتى تقوم بكتابة أي إجراء (مع العلم أنه عضو في صنف ما) فعليك أن تفعل ما يلي كما في السطر السابع عشر:

CODE

```
      اسم الصنف   اسم الإجراء
17    maths::print()
```

وكما تلاحظ فلقد فصلنا بين اسم الصنف واسم الإجراء بأربع نقاط وهذا ما عليك فعله عند كتابة أي دالة ضمن أي صنف.

بإمكانك فهم الكود من خلال التعليقات والشروحات المكتوبة ضمنه.

وقبل الانتقال إلى الفقرة القادمة فبامكانك قراءة الكود القادم وفهم ما يحويه وماذا يفعل.

أما عـن كيفيـة الوصول لأي دالة في البرنامج الرئيسـي فبإمكــانك فعـل ذلـك عـن طريق كتـابة اسم الكـائن المعلـن عــنه ثم اسـم الدالـة ويفصـل بينهـما نقطـة واحدة كما في السطر الثلاثين.

CODE

```
30    a.GetNum1Num2(i,j);
```

وقبل الانتقال إلى الفقرة القادمــة فبامكــانك قـراءة الكــود القـادم وفهـم مـا يحـويه وماذا يفعل.

CODE

```
1     #include <iostream.h>
2
3     class First // أطلقنا على هذا الصنف هذا الاسم
4     {
5      int d1; //يحـوي هذا الصنف على أربع متغيرات خاصـة الأول هـو البعد الأول من المصفوفة
6      int d2; // والآخر هـو البعد الثاني للمصفوفـة
7      int counterd1; //  أما الثالث فهـو عداد البعد الاول والذي سنستخدمـه في الدوال
8      int counterd2; // التكرارية وكذلك هناك عداد البعد الثاني
9      int **pArray; // وهـناك العنصر الخامس وهـو المصفوفة نفسها
10     public:
11     Enter ( int s1,int s2); //تستخدم هذه الدالة للوصول إلى العناصر الداخلية
12     putArray();// وظيفة هذه الدالة حجز الذاكرة للمصفوفة
13     Loop( );// تطلب هذه الدالة من المستخدم إدخال عنصاصر المصفوفة
14     print();// تستخدم هذه الدالة لطباعة عناصر المصفوفة
15    };
16    First::Enter(int s1,int s2)
17    {
18    d1=s1;
19    d2=s2;
20    }
21    First::putArray( )
22    {
23    pArray= new int *[d1];
24    for (counterd1=0 ; counterd1<d1;counterd1++)
25    pArray[counterd1]= new int [d2];
26    }
27    First::Loop()
28    {
29    for (counterd1=0;counterd1<d1;counterd1++)
```

```
30    for (counterd2=0;counterd2<d2;counterd2++)
31    cin >> pArray[counterd1][counterd2] ;
32
33    }
34    First::print()
35    {
36    for (counterd1=0;counterd1<d1;counterd1++)
37    for (counterd2=0;counterd2<d2;counterd2++)
38    cout << pArray[counterd1][counterd2] << endl;
39    }
40
41    int main ( )
42    {
43    First a;
44    int i,j;
45    cin >> i;
46    cin >> j;
47    a.Enter(i,j);// هنا نستدعي أول دالة والتي تقوم بالوصول إلى العناصر الداخلية للكائن
48    a.putArray();
49    a.Loop();
50    a.print();
51    }
```

حاول أن تفهم الكود السابق حتى تتأكد من أنك فهمت الأصناف والكائنات.

الأعضاء ومحددات الوصول:

أعضاء الصنف: هم جميع الدوال والمتغيرات التي تم تعريفها ضمن هذا الصنف.

ولضمان أنك تقوم بتطبيق فعلي للبرمجة الكائنية ولمبدأ الكبسلة خصوصاً فعليك أن تقوم بجعل جميع المتغيرات الأعضاء مكبسلين ، لا يوجد قاعدة عامة لذلك ، ولكن طبيعة البرمجة الكائنية تفرض عليك ذلك ، فجميع المتغيرات الأعضاء لن تطلب منها أنت أن تكون ظاهرة للعيان لأنها هي اللب الداخلي للصنف ، أو يكمن أن نعرفها على أنها **الحالة الداخلية للصنف** ، فالمتغيرات إذا تغيرت فستتغير طبيعة البرنامج الذي تقوم به ، أو المهمة التي يقوم بها هذا الصنف بعكس الدوال الأعضاء فيمكننا فهم الدوال الأعضاء على أنها هي المحرك للمتغيرات والمتغيرات يجب أن تبقى مخفية عن الجميع ما عدا هذه الدوال والتي تعرف كيف تتصرف معها . فحينما ترغب في أن يكون أحد المتغيرات الأعضاء لا يتغير أبداً مهما فعلت إلا وفق شروط معينة فستقوم بكبسلة هذا العضو المتغير وكتابة دالة تعرف كيف تتصرف مع هذا المتغير.

جميع الأعضاء المكبسلين لا بد أن يكون لهم محددات وصول ، فلنفترض أنك أردت طباعة قيمة أحد الأعضاء المتغيرات فلن تستطيع فعل ذلك بسبب أنه مكبسل ، ولفعل ذلك فلا بد أن تجعل لكل عضو متغير محدد وصول، وتقاليد

التسمية المتبعة (تقريباً في جميع العالم) لأسماء محددات الوصول هي أن تكتب كلمة Get ثم اسم العضو المتغير ، ومحدد الوصول يعيد قيمة العضو المتغير ، فمثلاً لو أردنا كتابة محدد وصول للمتغير العضو itsAge فسنكتبه هكذا:

CODE

```
1     GetitsAge() { return itsAge; }
```

صحيح أنه بإمكانك ابتكار طريقة لنفسك ، لكن لا بد أن تجعل برامجك مفهومة سواء لك ولغيرك فإذا أتيت بعد عدة أشهر لقراءة برنامج سابق فلن تفهم ما كتبته إلا بشق الأنفس وقد تقضي أسابيع لفعل ذلك وإذا اشتركت في كتابة أحد البرامج مع غيرك، فلا بد أن تكون هذه التقاليد (تقاليد التسمية) موحدة لديكم حتى يفهم كلاً منكم الكود الذي كتبه الآخر.

هناك أيضاً محدد وصول آخر وهو الدالة set وهي التي تقوم بتهيئة العضو المتغير أو مساواته بأحد المتغيرات ، وسنأتي بمثال على نفس نسق المثال السابق

CODE

```
1     SetitsAge(int x) { itsAge=x; }
```

كما ترى فإن نفس تقليد التسمية المتبع مع الدالة Get نتبعه هنا مع الدالة Set ، والذي تقوم به الدالة Set هو أنها تقوم بتغيير المتغير المكبسل حسب ما تريده أنت.

في النهاية محددات الوصول ليست قاعدة برمجية **بل هي رؤية أفضل لكتابة برامج أسهل للصيانة وللتطوير** ، فما هي الفائدة من كتابة برامج تشبه طلاسم السحرة ، وأنت الذي تحدد مدى حاجتك لهذه المحددات ، فبعض الأعضاء تريدهم أن يكونوا ثابتين ولا تريد طباعتهم أو تغييرهم أو فعل أي شيء فيهم.

تابع البناء:

سندخل الآن في أحد المواضيع المهمة ؛ كما تعلم فحينما تقوم بكتابة بيانات أي صنف فإنه وبقليل من التفكير ستستنتج أنه لا يمكنك وضع أي قيمة ابتدائية لأي من بيانات الصنف ؛ والسبب في ذلك أنك لا تقوم بحجز ذاكرة لهذا الصنف فكيف تحجز ذاكرة في الأساس لعنصر من عناصره . ومن أجل حل هذه المشكلة تم وضع دوال خاصة تسمى دوال البناء.

سنقوم الآن بتعديل المثال ما قبل السابق وسنجعله يعمل على تهيئة المتغيرات الأعضاء داخل الصنف

CODE

```
1     #include <iostream.h>

2     class maths
3     {
4     private:
5     float itsNum1;
6     float itsNum2;
7
8     public:
9     maths(float i,float j);
10    print();
```

```
11    };
12    maths::maths(float i,float j)
13    {
14    itsNum1=i;
15    itsNum2=j;
16    }
17    maths::print()
18    {
19     cout << "add:\n" << itsNum1+itsNum2 << endl;
20     cout << "subtract:\n" << itsNum1-itsNum2 << endl;
21     cout << "multiby:\n" << itsNum1*itsNum2 << endl;
22     cout << "divide:\n" << itsNum1/itsNum2 << endl;
23    }
24
25    int main ( )
26    {
27    float i,j;
28    cin >> i>>j;
29    maths a(i,j);
30    a.print();
32    return 0;}
```

كما تلاحظ في المثال الجديد فإن الأسطر 12-16 قد تغيرت وكذلك السطر 29 ؛ في السطر الثاني عشر وضعنا دالة جديدة لها نفس اسم الصنف وهذه ما تسمى بدالة البناء ؛ والتي يمكن تمييزها بأن لها نفس اسم الصنف الذي تنتمي إليه .. كما تلاحظ فإن دالة البناء تقبل الوسائط لكنها لا تعيد أي قيمة حتى القيمة void ؛ ومن الضروري أن نعلم أن لكل صنف تنشئه فإن المترجم ينشأ لك دالة بناء إفتراضية (في حال عدم كتابة دالة البناء) ، وفي حال كتابتك لدالة البناء فإن طريقة إنشاء كائن من الصنف تتغير حتى تصبح بالشكل الموجود في السطر التاسع والعشرون

CODE

```
      اسم الصنف   اسم الكائن   وسائط دالة البناء
29    maths    a   ( i , j );
```

وكما ترى فلقد أصبحنا نكتب وسائط دالة البناء عند إنشاء أي كائن ؛ ونكتبها بالتحديد بعد اسم الكائن الجديد وبين قوسين.

ادرس المثال السابق حتى تفهم موضوع دالة البناء بشكل أفضل.

قاعدة:
دالة البناء Constructor لها نفس اسم الصنف .. هذه الدالة لا تعيد أي قيمة حتى القيمة void
ولكن بإمكانها أن تأخذ أي عدد من المعاملات وبإمكانك زيادة تحميل مثل هذه الدوال.

تابع الهـدم:

بعـد أن تنتهي من الكائن الذي تعمل عليه فمن الضروري أن تقوم بهـدمـه أو حذفه حتى تحرر الذاكرة وبالتالي تزيد من السرعـة والأداء ؛ وهذا ما توفره لك دالة الهـدم ؛ بإمكـانك أن تحذف الأعضاء الذين لا تريدهـم مثل المؤشرات والمرجعيات وحذف الكائن بالكـامل. ادرس المثال القادم ؛ والذي لا يأتي إلا للتوضيح ليس إلا:

CODE

```
#include <iostream.h>
class First
{
public:
First(){cout <<"...class First has built"<< endl; }//دالة البناء
~First(){cout <<" class First has die";}//دالة الهدم
};
void main()
{
First m;
}
```

كما تلاحظ فإننا قمنا بكتابة دالة البناء والهـدم لطباعـة رسـائل معـينة حتـى نعرف متى أنشئت ومتـى انتهـت وسـتعرف أن دالـة البنـاء تـم تفعيلها حينمـا أعـلنا عـن كائن من الصنف first وأن دالة الهـدم تم تفعيلها حينما انتهينا مـن البرنامج.

قاعدة:
دالة الهـدم **Destructor** لها نفس اسم الصنف مسبوقاً بعلامـة (~).. هذه الدالة لا تعيد أي قيمـة حتى القيمة **void** وليس بإمكـانك تمرير أي معاملات لها لذلك فلن يكون بإمكـانك زيادة تحميل هذه الدالة.

متى يتم إستدعاء توابع الهـدم والبناء:

كما قلنا فإن إستدعاء دالة الهـدم يتم عـند إنشاء كائن ودالة الهــدم تـتم عــند تهـديم هذا الكائن.
إذا كان الكائن معرف بشكل عام أي خارج الدالة main() فإن دالة البناء هــي أول دالة يتم إستدعاؤها في البرنامج أما إذا كان الكائن معرف داخل أي دالـة ؛ فإن دالة البناء تستدعـي حسـب السـير الطبيعــي للبرنـامج ؛ ودالـة الهـدم يتم إستدعـاؤها عـندما يصبح الكائن خارج مدى الرؤية.

التوابع الأعضاء السطرية:

حينمـا تقـوم بتعريـف أي دالـة ضـمن كتلـة تعريـف الصـنف فإنـه يصـبح دالـة سـطرية (inline) حتـى من دون كتابة الكلمة المفتاحية الدليلية inline .

قاعدة:
الدوال الأعضاء السطرية **inline function** هـي المعرفـة داخل كتلة الصنف ؛ أما الدوال الأعضاء غير السطرية **non-inline function** فهـي معرفة خارج الصنف ومصرحـة داخل كتلة الصنف ولإجبار المترجم على التعامل مع أي دالة عضو على أنها دالة سطرية معرفة خارج كتلة الصنف فبإمكـانك كتابة الكلمة المفتاحية **inline** ضمن تعريف الدالة.

المؤشر this:

حتى تستطيع التمكن من مزايا البرمجة الكائنية التي تمنحها لك C++ فعليك أن تستفيد من المؤشرات والمرجعيات بأقصى طريقة ممكنة بالرغم من صعوبتها وخطورتها الشديدة (ارجع إلى مواضيع المؤشرات في هذا الكتاب إن لم تكن مفهومة لديك) .

يحتوي كل كائن على مؤشر اسمه this ، هذا المؤشر يشير إلى الكائن نفسه حتى يستطيع استدعاء النسخة الصحيحة من التوابع أو المتغيرات الأعضاء.

لنفرض أن لدينا صنف اسمه Test ولدينا كائنان آخران اسمهما a و b ، فحينما تقوم باستدعاء أحد التوابع التي تعالج أحد المتغيرات الأعضاء فإن المترجم لن يعرف أي نسخة من المتغيرات تقصد هل هي للكائن a أو الكائن b ، لذلك يتم تمرير المؤشر this إليه ، وهذا المؤشر يمنع المترجم من الخلط بين الكائنين وبالتالي التعامل مع النسخة الصحيحة من المتغيرات والتوابع الأعضاء.

لاحظ أن مؤشر this مخفي عنك وسيقوم المترجم بوضعه نيابة عنك في حال لم تقم به ، هناك بعض الاستخدامات للمؤشر this وهي كثيرة ستجد بعضاً منها في الوحدة القادمة.

سنقوم بكتابة مثال يوضح لك عمل المؤشر this ، انظر إلى هذا المثال:

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. class stud{
5. public:
6.     void address(){cout << this;
7.     }
8. };
9.
10.    int main()
11.    {
12.          stud a,b,c;
13.          cout << "The address of a\t" ;
14.          a.address() ;
15.          cout << endl << "The address of b\t" ;
16.          b.address() ;
17.          cout << endl << "The address of c\t" ;
18.          c.address() ;
19.          cout << endl;
```

```
20.
21.         return 0;
22.    }
```

قمنا بالإعلان عـن صنف هـو stud ولا يحـوي سوى تـابع واحـد يقـوم بطباعـة محتويات المؤشر this ، قمنا أيضاً بالإعلان عـن ثلاث كائنـات من نفس الصنف ثم قمنا بإستدعاء التابع address لكل كائن ، لاحـظ أن كـل نـاتج مختلـف عــن الكائن الآخر.

الأعضاء السـاكنـة Static Members:

تختلف هذه الأعضاء في طبيعتها عـن جميع البيـانات الأخــرى ؛ فلـو افترضنـا مثلاً أن لديك صنف اسمـه (Test1) ويوجد في هذا الصنف عضو متغير اسمـه (i) وقمت بإنشاء كائنين من ذلـك الصـنف .. فإنـك بـديهياً سـتعتقد أنــه أصـبح هــناك نسختــان مـن العضـو i ؛ الأولـى تابعــة للكـائن الأول والثانيــة تابعــة للكائن الثاني وهذا الاعتقاد صحيح ، إلا أنـه لا يمكن تطبيق هذا الشـيء على الأعضاء السـاكنـة فإذا قمت بالتصريح عـن عضو علـى أنــه سـاكـــن فعليـك أن تعلم أنـه عبارة عـن نسخـة واحدة لجميع الكائنـات فمثلاً لو افترضنا أنك قمت بإنشاء صنف اسمـه Arrays ويحتـوي على عضو متغيـر ســاكـــن اسمـــه A ثـم بعـد ذلك أنشئت أكثر مصفوفة تحـوي أكثر من 100 عـــنصر مـن نمـط الصـنف Array فإن هذا لا يعـني أنه يوجد 100 متغير A بل يوجد فقط متغيـر A ينتمـي إلى جميع أعضاء الصنف ولو تغير هذا العضو في أي كائن فإنـه سـيتغير فـي البقيـة جميعها وهـكذا.

قد تتساءل عـن الفائدة العـملية لهذا المتغير الساكـن إلا أنـه لـه فوائد جمـة ستتعرف عليها لاحقاً في هذا الكتـاب ؛ أما الآن فدعـنا نأخذ مثالاً عـملياً على هذا الموضوع:

CODE

```
1      #include <iostream.h>

2      class First
3      {
4      public:
5      static int counter;//.هذا هو تصريح المتغير الساكن
6      First( )
7      {
8      counter++;}
9      getCounter() {return counter;}
10     };

11     int First::counter=0;//هذا هو تصريح العضو المتغير الساكن

12     void main()
13     {
14     First a;
```

```
15      First b;
16      First c[60];
17      cout << a.getCounter();// هنا نطبع القيمة التي تعيدها الدالة وليس الدالة نفسها
18      }
```

كما تلاحظ فلقد قمنا بالتصريح عن صنف اسمه First وقمنا بإنشاء أكثر من 62 كائن من هذا الصنف. وكما تلاحظ فإن العضو الساكن الوحيد هو counter والذي تقوم دالة البناء التابعة للصنف بزيادته مرة واحدة عند كل إستدعاء لها ؛ في 17 قمنا بطباعة الدالة العضو getCounter والتي هنا تابعة للكائن a (وليس لآخر كائن في المصفوفة c) وجاءت النتيجة بأن قيمة counter هي 62 وهو عدد الكائنات الموجودة في البرنامج.
قد تستغرب من وجود السطر الحادي عشر ، بالرغم من أنك تعلم أنه لا يمكنك تهيئة أي عضو داخل تصريح صنف إلا في دوال البناء أو أي دالة أخرى إلا أن الحال مختلف بالنسبة للأعضاء الساكنة فحينما يقوم المترجم بترجمة البرنامج فإنه يحجز ذاكرة للعضو الساكن قبل أن يحجز لأي كائن (حسب السير الطبيعي للبرنامج) ؛ إذا لم تقم بتعريف العضو الساكن (إذا ألغيت السطر 11 من المثال السابق مثلاً) فسيعطيك المترجم رسالة خطأ أو ربما الرابط وليس المترجم ؛ فيجب عليك ألا تنسى تعريف هذه الأعضاء.
هذا بالنسبة للمتغيرات الأعضاء الساكنة داخل أي صنف وفي الحقيقة فإن هذا الأمر ينسحب عموماً إلى الدوال الأعضاء الساكنة.

التوابع الأعضاء الساكنة Static Members Functions:

التوابع الأعضاء الساكنة تمكنك من الوصول إلى المتغيرات الأعضاء الساكنة الخاصة ليست العامة حتى دون الإعلان عن أي كائن من نفس الصنف، قد تتساءل عن ماهية الفائدة ، ولكن لما لا تجعل هذه الميزة أقصد البيانات الساكنة إحدى معارفك ومعلوماتك البرمجية وصدقني سيأتي اليوم الذي تحتاج فيه إليها.

CODE

```
1       #include <iostream.h>

2       class First
3       {
4
5       static int counter;
6       public:
7       static getCounter() {return counter;}
8       First( )
9       {
10      counter++;}
11
12      };
13
```

```
14      int First::counter=15;
15      void main()
16      {
17              cout << First::getCounter()<< endl;
18      }
```

لقد جعلنا من الدالة getCounter( ) دالة وصول عامة ساكنة وبالتالي فبإمكاننا الحصول على فوائدها دون حتى الإعلان عن أي كائن من الصنف First ، وتصريح الدالة في السطر 7 يدل على أنها أصبحت دالة وصول ساكنة.

الإحتواء أو التركيب:

هذا الموضوع يعتبر أحد أهم المواضيع ، وبالرغم من أهميته فليس هناك ما يدعو إلا اعتباره موضوعاً صعباً للغاية ، ولكن حتى تصل لأقصى ميزات الإحتواء فعليك أن تجعل من واجهة الصنف الذي تريد إحتواؤه واجهة كاملة أي يجب أن يكون لكل متغير عضو، دالة get( ) و دالة set( ) خاصة به عدا بعض الأعضاء الذين يعتبر التعامل معهم خطيراً للغاية ، يعرف الإحتواء على أن تركيب أحد الأصناف يعتمد على صنف آخر ، فمثلاً إذا كان لدينا الصنف سيارة فإن الصنف المحرك يعتبر أحد الأصناف الرئيسية في تركيب الصنف السيارة ، لذلك فإن الصنف محرك يعتبر محتوى في الصنف السيارة ، يمكن وصف العلاقة بين الصنفين بأنها (يمتلك) أي أن الصنف السيارة يمتلك الصنف المحرك ، يعتبر هذا الكلام ضرورياً للغاية حينما تصل لمواضيع الوراثة وكيف تفرق بين العلاقات بين الكائنات أهي علاقة توارث أم تركيب واحتواء . عموماً وضعنا أحد الامثلة على التركيب وهو الصنف Data الذي يحتويه الصنف Student ، ستلاحظ أن الصنف Data سيكون أحد الأعضاء المتغيرات الخاصة في الصنف Student ، ذلك لا يعني أنه بإمكان الصنف Student الوصول إلى المتغيرات الخاصة في الصنف Data وحتى يصل إليها فعليه الإعتماد على محددات الوصول ن لذلك فهناك فائدة كبرى لمحددات الوصول في أي صنف تقوم بكتابته ، ليس هذا المثال مثالاً عظيماً بل هو مثال حتى تفهم أحد العلاقات بين الأصناف وهي التركيب أو الإحتواء:

CODE

```
1       #include <iostream.h>
2
3       class Data
4       {
5       double itsAvreg;
6
7       public:
8               getItsAvreg(){return itsAvreg;}

9               setItsAvreg(double x){itsAvreg=x;}
10
11      };
```

```
12
13    class Student
14    {
15          Data itsData;
16    public:
17          getItsAvreg(){ return itsData.getItsAvreg();}
18          setItsAvreg(double x){itsData.setItsAvreg(x);}
19    };
20
21    int main()
22    {
23          Student a;
24          a.setItsAvreg(98);
25          cout << a.getItsAvreg();
26    }
```

تذكر أن هذا الموضوع يعتبر أحد المواضيع المهمة ، بالرغم من قصره لذلك إذا لم تفهم فأعد قراءته من جديد ، وحاول تطبيق ما قرأته على الأمثلة القادمة.

اللغة Smaltalk والكائنات:

أول لغة كائنية ناجحة ظهرت في الوجود ، هي لغة smaltalk ، سنقوم في هذه الفقرة بأخذ مبادئ هذه اللغة ومجاراتها هنا في لغتنا السي بلس بلس لفهم أفضل لما تعنيه الكائنات في البرمجة.

حسب مؤلف لغة smalltalk ، فإنها تقوم على خمس مبادئ وهي مهمة هنا في حالتنا إذا ما أردنا النجاح في البرمجة الكائنية:

1- كل شيء هو كائن :
2- البرنامج عبارة عن كائنات تتفاعل مع بعضها بواسطة إرسال الرسائل:
3- كل كائن يملك ذاكرة خاصة به مبنية على الكائنات الأخرى.
4- لكل كائن نوع من البيانات (أي صنف).
5- جميع الكائنات من نفس النوع تتفاعل بواسطة نفس الرسائل.

إذا ما فكرت جيداً في هذه المبادئ الخمسة فسوف تجد أنها تتكلم عن الكائنات وليس الأصناف بالتالي فإن عليك أن تتذكر أن البرمجة الكائنية قائمة على الكائنات وليس الأصناف.

لكل كائن واجهة:

يعتبر هذا الموضوع أحد المبادئ المهمة حينما تقوم بصنع صنف حتى تستفيد من الكائنات ، الواجهة هي البيانات العامة للكائن ، ويجب عليك أنت صنع كائن جيد حيث أن هذا هو الاتجاه في البرمجة الكائنية ، دعنا نفكر قليلاً في أهمية هذه الواجهة (البيانات العامة) ، أولاً أنها هي الطريقة الوحيد حتى يتفاعل هذا الكائن مع الكائنات الأخرى أو مع البرنامج توابعاً كان أو كائنات أو متغيرات ، ثانياً البيانات العامة يجب ألا تكون هي

اللب الرئيسي للكائن لأنها إن كانت كذلك فسيستطيع المستخدم العبث بمحتويات هذا الكائن ، ليس عن قصد بل عن خطأ ، لأنه لا يعرف ما هي الاشياء المهمة لهذا الكائن التي هي الواجهة ، فمثلاً إذا ما أراد المستخدم (مستخدم الصنف) تغيير إحدى البيانات فسيكون التغيير آمنا بواسطة التوابع set و get . من هنا يجب عليك الفصل بين الواجهة والمعالجة ، المعالجة يجب أن تكون داخلية وليست خارجية . فمثلاً إذا قمنا بكتابة صنف طالب فحينها يجب علينا إخفاء البيانات المهمة والتي نود معالجتها .
إخفاء المعالجة في أغلب الأحيان يتطلب منك جعل البيانات الفعلية للأصناف مثل صنف طالب مخفية (درجة الطالب ، معدل الطالب ، عمر الطالب) مخفية عن العالم الخارجي وبالتالي فأنت ستسمح فقط للتوابع الأعضاء بمعالجة هذه البيانات المهمة وبالتالي فأنت قمت بإخفاء المعالجة نهائياً.
قد تستغرب من هذا الكلام وحول فائدتها لكن عليك أولاً أن تفصل بين مفهوم صانع الصنف ومستخدم الصنف ، إذا كنت في شركة فلن تقوم أولاً بكتابة أصناف لبرامجك بل ستبحث عنها ممن سبقوك وبالتالي فبإمكانك توسعتها عن طريق الوراثة وتعدد الاوجه أو إبقائها على ماهي واستخدامها وبالتالي زيادة الإنتاجية ؛ وحينما تقوم أنت بإستخدام هذه الأصناف التي كتبها من سبقوك فستهتم أكثر وأكثر بالواجهة لهذا الصنف فمثلاً في صنف طالب لن تهتم كيف قام هذا الصنف بحساب النسبة المئوية ليس لأنها كيفية حسابها معروفة بل لأنك تتوقع أن هذا الصنف جيد بما فيه الكفاية حتى لا تقوم أنت بالتأكد مما يفعله ... قد تتساءل الآن عن خطورة هذا الإجراء ولكن هذا ما نود التأكيد عليه قم بالتركيز على ميدان المشكلة التي تقوم بحلها وليس على التفاصيل الكائن السابق سيكون قد مر على مئات الاختبارات للتأكد من صلاحيته وفعاليته وبالتالي فأنت عليك التركيز على كيفية استخدام هذا الصنف في برامجك وليس على تفاصيل الصنف ، ووسيلتك لتحقيق ذلك هي واجهة هذا الصنف.

مثال واقعي :

ربما لم تفهم الكلام السابق ولا حرج في ذلك فهو غامض على أغلب المبرمجين الجدد ، دعني الآن أجلب لك مثالاً من الحياة الواقعية وهو عن الحاسبات ، الواجهة الرئيسية لكل حاسب هي لوحة المفاتيح والشاشة والفأرة وبعض الأجهزة الاخرى ولكن الثلاث هذه هي الأهم ، ألا ترى معي أن الحاسب الذي قبل عشر سنوات هو نفسه حالياً وأنه لا اختلاف بينهما ، الاختلاف الوحيد الكبير هو داخل هذا الحاسب أو العمليات التي تجري في الحاسب ، أنت كمستخدم لهذا الحاسب لا تهتم أبداً بهذه التفاصيل لأنه في الأساس ليس مطلوباً منك أن تهتم بل كل ما عليك أن تهتم به هو استخدام هذا الصنف أو هذا الحاسب. هذا المثال هو ما أريدك أن تقوم بتطبيقه في حياتك البرمجية على مستوى الاصناف ، يجب عليك أن تقوم بتجريد وفصل المعالجة عن الواجهة ، لا تجعل إحدى العمليات الداخلية للصنف بيانات عامة ، دعني أخبرك عن خطورة هذا الإجراء في مثال الحاسب ، لنفرض أن إحدى الشركات قامت بصنع حاسب ، وقامت أيضاً بإخراج إحدى المعالجات من صندوق الجهاز وقالت لجميع مستخدمي هذا الحاسب إذا أردت تشغيل الحاسب لساعة واحدة فعليك أن تقوم بشبك سلكين فقط أما إذا أردت إيقاف الجهاز فعليك قطع ثلاث أسلاك وغير هذا من الكلام المفصل حينها ستكون هذه الشركة جعلت إحدى العمليات الداخلية تصبح واجهة بعد عشر سنوات قامت هذه الشركة بتطوير منتجها وبالتالي فستقوم الآن بتعديل وتطوير جميع العمليات الداخلية داخل الحاسب والتي سيكون من ضمن التطوير ذلك المعالج وحينها فسيتغير كل الكلام السابق للمستخدمين ، إذا ما أردت تشغيل الجهاز فعليك شبك سلك واحد بقوة 250 فولت وإذا أردت إيقاف الجهاز فعليك تخفيف الجهد إلى 10 فولت !!!! ، حينها سيقوم جميع

الزبائن برمي أجهزة هذه الشركة إلى الأبد ، وهذا ما عليك تجنبه حينما تقوم بتطوير الصنف الذي تريد ، افصل بين المعالجة والواجهة.
حتى تفهم جميع كلامي السابق بشكل أفضل قم بكتابة كود بلغة السي وقم بكتابة نفس هذا الكود بلغة السي بلس بلس ولكن باستخدام الكائنات، حاول بعد ذلك تطوير الكودين بعد شهر من الآن حينما تكون قد نسيت محتوياتهما ، وانظر إلى الفرق بينهما ففي الكود الأول ستضطر إلى إعادة التفكير من جديد في نفس المشكلة أما في الكود الآخر فلربما لن تقوم بالتفكير أو حتى إضافة سطر كودي جديد بسبب تطبيقك للمبادئ والمفاهيم السابقة.

::أمثلة تطبيقية::

مثال1/

قم بكتابة صنف يقوم بتهيئة مصفوفة ثنائية متغيرة الحجم.. مع تضمين هذا الصنف دوال الهدم والبناء وتستطيع عبر هذا الصنف تصفير القطر الرئيسي للمصفوفة.

الحل:

سنقوم بتصميم هذا الصنف كما يلي:

- سنعتبر أن المتغيرات الأعضاء الخاصة هم: الصف والعمود وعدادين اثنين سنستخدمهم لإدخال عناصر المصفوفة وطباعة هذه العناصر .
- سنقوم بإنشاء هذه الدوال ونعتبرها دوال عامة: دالة البناء والهدم ودالة تقوم بتصفير القطر الرئيسي للدالة ودالة تمكن المستخدم ودالة أخرى لطباعة عناصر المصفوفة.
- دالة البناء ستقوم بتهيئة المصفوفة ؛ فيما نمكن المستخدم من الإختيار بين أن يطبع عناصر المصفوفة مع تصفير القطر الرئيسي أو لا.
- بإمكاننا دمج دالة تمييز عناصر المصفوفة ودالة الطباعة ودالة تصفير القطر الرئيسي في دالة واحدة.

الكود:

CODE

```
#include <iostream.h>

class Array
{
float **arrays;
char choice;
int itsD1;
int itsD2;
int itsD1Count;
int itsD2Count;
public:
      Array(int ,int);
      ~Array();
      Bigfunction();
      print();
```

```
        Enter();
};

Array::Array(int i,int j)
{
itsD1=i;itsD2=j;
arrays=new float*[itsD1];
for (itsD2Count=0;itsD2Count<itsD2;itsD2Count++)
arrays[itsD2Count]=new float [itsD2];

}
Array::~Array()
{
delete [] arrays;
}
Array::Enter()
{
cout << "Enter the memeber of Array" << endl;
for ( itsD1Count=0;itsD1Count<itsD1;itsD1Count++)
5           for (itsD2Count=0;itsD2Count<itsD2;itsD2Count++)
3                   {
3                   cout <<"Enter the member:\t" << endl;
2                   cin >> arrays[itsD1Count][itsD2Count];
1                   }
}
Array::Bigfunction()
{
if (itsD1==itsD2)
{
1           cout << "Do you want to make the main Rayon  Zero[y/n]";
2           cin >> choice ;
1                   if (choice=='y')
12                  {
2                   for ( itsD1Count=0;itsD1Count<itsD1;itsD1Count++)
12                  {
1                   arrays[itsD1Count][itsD1Count]=0;
1                   }
12                  print();
1       }
1                   else
12                          print();

}
```

```
else
print();
}
        Array::print()
        {
        cout << endl;
        for ( itsD1Count=0;itsD1Count<itsD1;itsD1Count++)
        {
                for (itsD2Count=0;itsD2Count<itsD2;itsD2Count++)
12              {

1               cout << arrays[itsD1Count][itsD2Count];
2               cout << "\t";
1               }
1               cout << endl;

1               }
1               }

12      int main()
        {
        int x,y;
        cout << "enter d1:\t ";cin >> x;
        cout <<"enter d1:\t ";cin >>y;
        Array One(x,y);
        One.Enter();
        One.Bigfunction();
        return 0;
        }
```

مثال2/

قم بكتابة صنف يشبه محرر النصوص (النوت باد) يستطيع المستخدم عند ضغطه على حرف (p) أن يخرج من المحرر ثم يعرض عليه البرنامج عدد الأحرف التي كتبها.

الحل:

سنقوم بتصميم هذا الصنف كما يلي:

- محرر النصوص الذي سنقوم بإنشاءه سيكون سهلاً للغاية ولن يكون معقداً وإن كان بإمكانك تطويره حتى يصبح محرر نصوص مقبولاً.

- محرر النصوص يقوم بقبول أكثر من 4000 حرف تستطيع إدخاله ويقوم بتخزين كل ما تكتبه أيضاً مباشرة ؛ إلا أنه لن يقوم بتخزينه في ملف.
- عندما تكتب الرقم 1 فإن محرر النصوص يخرج من البرنامج ويخبرك بعدد الأحرف التي أدخلتها.

الكـود:

CODE

```
1   #include <iostream.h>
2
3   class notepad // سنطلق على هذا الكائن اسم النوت باد
4   {
5   int index1,index2;// هذه المتغيرات ستستخدمها في دوارة الإدخال حتى يعرف الحاسب إلى أين وصل
6   char One[200][200];// سنخزن في هذه المصفوفة كل ما يكتب في هذا المحرر
7   int charactor;// هذا المتغير هـو الذي يحسب عدد الأحرف المدخلة
8   public:
9   notepad ();// إجراء البناء يقوم بتهيئة المتغيرات المهمـة بالقيمة صفر
10  HowMany();//تقوم هذه الدالة بحساب عدد الأحرف وأيضاً تسمح للمستخدم بإدخال ما يريد في المحرر
11  display();// يظهر هذا الدالة عدد الأحرف المدخلة
12
13  } ;
14  notepad:: notepad ( )
15  {
16  charactor=0;
17  }
18  notepad::HowMany()
19  {
20  cout <<"\n";
21  for (index1=0 ;index1<200;index1++ )
22  {
23  for ( index2=0;index2<256;index2++  )
24  {
25  cin >> One[index1][index2];
26  if (One[index1][index2]=='1')// هنا يقوم البرنامج بتمييز إذا كان المدخل العدد 1 فإنه يخرج
27                                // عن محرر النصوص ويذهب إلى الدالة التالية في تنفيذ البرنامج
28  return 0;
29  else charactor++;//إذا لم يكن المدخل هـو الرقم 1 فإن البرنامج يزيد من قيمـة هذا المتغير
30  }
31  }
32  }
33  notepad::display()
34  {
35  cout << "The number of char you made it is\t" << charactor << endl;
```

```
36
37      cout <<"\a"  ;//يقوم البرنامج بإصدار صوت تنبيهي عندما يظهر ناتج عدد الأحرف المدخلة//
38      }
39
40      void main()
41      {
42      notepad first;
43      first.HowMany();
44      first.display ();
45      }
```

مثال3/

هل تتذكر المتسلسلة الحسابية fibancci **والتي دائماً ما تكثر الكتب من ذكرها في أمثلتها ، عموماً فإن هذا الكتاب لن يشذ عن القاعدة إلا أننا الآن سنتعامل مع هذه المتسلسلة كصنف أي يجب أن نستفيد من هذا الصنف ولا يجب علينا أن نتركه يذهب سدى هكذا ، لم نجعل من الصنف** fibancci **صنفاً خارقاً لذلك فسنترك لك بقية المميزات حتى تكملها أنت بنفسك ، علماً أنه لا يمكنك التغاضي عن الميزات الحالية المقدمة.**

الحل:

سنقوم بتصميم هذا الصنف كما يلي:

- **سنطلق على هذا الصنف اسم** Fibancci **حتى يكون اسمه مماثلاً للغرض من الصنف.**
- **الغرض من هذا الصنف هو إيجاد المتسلسلة الحسابية وتخزينها كاملة حتى نستطيع الإستفادة منها.**
- **الأعضاء المتغيرات الخاصة هم** first **و** second **و** third **و** max **، المتغيرات الثلاث هم المتسلسلة الحسابية ، فيما المتغير** max **هو أكبر عدد تصل إليه المتسلسلة.**
- **لا داعي لأن أذكرك بما تعنية المتسلسلة الحسابية** fibancci **، حيث أنها تعني أن أي عدد هو مجموع العددين الذين قبلاه عدا أول عددين حيث أنهما يساويان الواحد ؛ شكل المصفوفة هكذا:**

1 1 2 3 5 8 13 21 34 55

- **سنقوم بإنشاء متغيران عضوين جديدين ، الأول هو سنطلق عليه مسمى** time **حيث يحسب عدد المرات التي يقوم بها بعملية الجمع حتى يصل إلى أصغر عدد ممكن من العدد الذي أدخله المستخدم ، أما المتغير الثاني فهو الأهم وسنترك لك مسؤولية تطويره وهو عبارة عن مصفوفة تخزن فيها المتسلسلة الحسابية.**

CODE

```
1. #include <iostream.h>

2. class fibancci
```

```
3. {
4. /* الخاصـة        المتغيرات */
5. int first,second,third;
6. int *array,max,times;
7. SetTimes()      /* تقوم بتهيئـة المتغير    دالة عضوة خاصـــة */
8. {
9. for (times=0;second<max;times++)
10.     {
11.     third=second+first;
12.     first=second;
13.     second=third;
14.     }
15.     third=second=first=1;
16.     }
17.     public:          /*البنـاء    دوال  */
18.     fibancci():times(1),first(1),second(1),max(50){Array();}
19.     fibancci(double x):times(1),first(1),second(1),max(x)
   {Array();}
20.     /*     الوصول  محددات */
21.     GetTimes()
22.           { return times;}
23.     Array()       /*    دالـــــة  أهم  */
24.           {
25.                 SetTimes();
26.                 array=new int[times];
27.                 for(int i=0;second<max;i++)
28.                 {
29.                       array[i]=first;
30.                       third=second+first;
31.                       first=second;
32.                       second=third;
33.
34.                 }
35.                 array[i+1]=first;
36.                 cout << endl;
37.           }
38.           printfibancci()   /* دالة لعرض المتسلسلة الحسابية */
39.           {cout << endl;
```

```
40.                 for (int i=0;i<times;i++)
41.                 cout << array[i] << "\t";
42.           cout <<"\t" << array[i+1] << endl;
43.           }
44.
45.     };
46.
47.     void main()
48.     {     double j;
49.           cin >> j;
50.           fibancci a(j);
51.           a.printfibancci();
52.           a.GetTimes();
53.     }
```

بالرغم من أننا قمنا بتسهيل طريقة إستخدام هذا الصنف إلا أنها لا تصل إلى ما هو مرجو منها ، قد يكون ممكناً فعل ذلك حينما نصل إلى وحدة (اصنع أنماط بياناتك بنفسك)، المهم في هذا الموضوع هو أن هذا الصنف يتألف من 4 دوال غير دوال البناء ، سنقوم بشرح هذا الكود:

لا تأخذ أي بارامترات بالنسبة للدالة الأولى أما الدالة الثانية في السطر 19 فهي تأخذ بارامتر واحد ، تقوم الدالتين جميعهما بتهيئة الثلاث العناصر الرئيسية حيث العنصران first و second بالقيمة 1 أما المتغير max ففي الدالة الأولى تتم تهيئته بالقيمة 50 أما الدالة الثانية فتقوم بتهيئة المتغير max بالعدد الذي قام مستخدم الصنف بتمريره. جميع دالتي البناء تستدعي الدالة Array( ) .

في بداية تنفيذ الدالة Array( ) يتم تنفيذ الدالة SetTimes( ) .

نظراً لخطورة التعامل مع المتغير times لأنه هو أهم متغير تقريباً في الصنف فقد جعلنا التعديل على هذا الصنف يكون من داخل المتغيرات والعمليات التي تحدث لها وليس بواسطة المستخدم لذلك جعلناه عضواً خاصاً ، لا أعتقد أن هناك شيئاً مهماً في هذه الدالة عدا الشرط الذي تفرضه الدوارة for في السطر 9 ، حيث أن شرطها الوحيد هو ألا يتجاوز العدد الثاني من المتسلسلة العدد الذي أدخله المستخدم وهذه هي الحالة الوحيدة التي بإمكانك إيجاد بها المتسلسلة بواسطة الدوارة for ما يهمنا الآن هو أن الدوارة for تحسب عدد الأرقام التي تم تنفيذها إلى الآن حتى تصل إلى العدد الذي أدخله المستخدم (أو بمعنى أدق أقل رقم من العدد الذي أدخله المستخدم) وتتوقف ثم ينتقل التنفيذ إلى السطر 15 حيث يتم تنظيف المتغيرات الثلاث للمتسلسلة من جميع آثار دوارة for حيث أن قيمها الآن أصبحت متغيرة ولم تكن مثل السابق ، يخرج بعدها التنفيذ من الدالة setTimes( ) ويرجع إلى الدالة Array( )، وتذكر أننا إلى الآن ما زلنا في تنفيذ دالة البناء.

في السطر 26 تقوم الدالة بإنشاء المصفوفة array التي ستحوي جميع أعداد المتسلسلة الحسابية ثم ينتقل التنفيذ إلى السطر 27 ، حيث يتم تخزين جميع الأعداد في المصفوفة الجديدة عدا آخر رقم في المتسلسلة إلا أن السطر 35 يتدراك هذا الأمر ولا أدري إلى الآن لماذا يحدث هذا ؟ ، أما كيف توصلت إلى هذا الحل فهو عن طريق إختبار الصنف أكثر من مرة وتجريب الأمثلة عليه للتأكد أنه يعمل بخير ، بالنسبة للدالة الرابعة والأخيرة فلا جديد فيها فهي فقط تقوم بعرض المصفوفة كاملة على الشاشة.

اصنع أنواع البيانات التي تريدها
Make your own Data Types

بداية:

بالرغم من حماسية الموضوع الذي اخترته لهذا الفصل ، فهو العنوان الوحيد الذي يجمع بين المواضيع التي سيتناولها هذا الفصل .. بإمكانك بعد أن تنتهي من هذا الفصل أن تنشئ أنواع البيانات التي تريدها فقد تنشئ نوعاً جديداً تختار له اسمك اسماً له مثل int أو float وقد تجعل له ميزات أعظم وأكبر من ميزات أنواع البيانات العادية كأن تجعله يستطيع التعامل مع الأعداد المركبة أو التخيلية وبإمكانك أيضاً أن تجعل علامة الجمع + بدلاً من أن تجمع عددين تقوم بطرحهما أو الضرب أو القسمة أو أي عملية حسابية أخرى تريدها... وبالرغم من أن هذا الفصل بالفعل يتناول هذه المواضيع (والتي قد تعتبرها أنت شيقة) فإنه يجب عليك المرور عليه لأن هذه المواضيع نتناولها أيضاً في مواضيع أخرى (غير صناعة أنماط بيانات جديدة) وخاصة موضوع التحميل الزائد والدوال الأخرى وغير ذلك من المواضيع المهمة.

حتى تتمكن من هذا الفصل جيداً فعليك الرجوع إلى فصل الفصائل والكائنات والمؤشرات أيضاً لأنها جميعها ضرورية لهذا الفصل.

** مقدمة في التحميل الزائد للتوابع:

لن نخوض كثيراً في هذه المقدمة لأنه يفترض أنك تعلمتها (أثناء دراستك للتوابع) وهذه المقدمة ما هي إلا فقط تذكير لما درسته سابقاً.

التحميل الزائد للتوابع هو عبارة عن دالتين أو أكثر تحمل نفس الاسم إلا أنها تختلف في عدد الوسائط أو أنماطها أو حتى ترتيبها.

وحتى نفهم موضوع التحميل الزائد فلنفترض أن لدينا عاملان اثنان الأول نجار والآخر حداد وأنك أنت المهندس. وأردت مثلاً أن تصنع باباً خشبياً فإنك (أي المهندس) لن تهتم بالتفاصيل وستقوم بتسليم المهمة لرئيس العمال (والذي هو في هذه الحالة المترجم) وهو سيقوم بتسليم المهمة للعامل المناسب والذي هو النجار. وبالرغم من تشابه الأسماء بين النجار والحداد (يشتركان في اسم العامل) فإن رئيس العمال لن يخطيء مثلاً ويقوم بتسليم المهمة للحداد (لماذا؟) لأنه يعرف النجار والحداد مالذي يستطيعان فعله ومالذي لا يستطيعان فعله وفي حال مثلاً أن المهندس طلب من رئيس العمال صنع قواعد مناسبة للمنزل فإن رئيس العمال (الذي هو المترجم) سيصدر خطأ ويخبرك بأنه لا يوجد لدينا مثل هذا العامل. حتى نفهم أكثر فلننظر إلى الباب الخشبي على أنه أحد الوسائط. بالتالي من هو العامل الذي يستطيع إستقبال مثل هذا الوسيط؟!.

أعتقد أن الموضوع إلى هذا الحد كافي وسنقوم الآن بتزويدك بأحد الأمثلة:

CODE

```
1    #include <iostream.h>
2
3    plus (int x,int m)//سنقوم بتحميل هذه التابع لتستقبل وسائط أخرى
4    {
```

```
5       return x+m;
6       }
7
8       plus (long x,long z)
9       {
10      return x+z;
11      }
12
13      main()
14      {
15      int a=10,b=20,c=0;
16      long d=30,e=40,f=0;
17
18      c=plus (a,b);
19      f=plus(d,e);
20      cout << c << endl << f;
21      //f=plus (a,d);   لن يتم تنفيذ هذا الدالة بسبب أنه خاطيء لذلك وضعنا قبلها علامة التعليق أو التوثيق
22
23      return 0;
24      }
```

لقد قمنا بزيادة تحميل التابع plus ( ) ففي المرة الأولى جعلناها تستقبل متغيرين من النمط int ثم تجمعهما وفي المرة الثانية جعلنة تستقبل متغيرين من النمط long وتقوم بجمعهما. وكما تلاحظ ففي السطرين 18 و 19 فلم نهتم إلا باسم التابع ولم نهتم بأي تفاصيل أخرى لم نهتم أصلاً بما يوجد داخل التابع plus( ) عدا بعض المعلومات البسيطة عن وسائطه ومن أجل ذلك وبسبب السطر 21 فلن تتم ترجمة المثال السابق إذا ألغينا علامة التوثيق لأن التابع plus لا تستطيع التعامل مع هذا النوع من الوسائط. حتى تفهم أكثر طبق مثال العمال والمهندس الذي ذكرناه في الصفحة السابقة على هذا المثال (أو أرجع للشكل في بداية هذا الموضوع).

قد تتساءل عن فوائد التحميل الزائد وهذا ما سنحاول الإجابة عليه.

** دوال البناء وزيادة التحميل:

لقد جعلنا عنوان هذه الوحدة هي اصنع أنماط بياناتك وحتى يكون نمط البيانات الذي نصنعه جيداً ويعتمد عليه ؛ فلا بد أن نجعل منه شيئاً بسيطاً لا صعباً وحتى يتحقق هذا الشيء أو حتى تفهم ما أقصده فدعنا نلقي نظرة على هذا الكود:

CODE

```
1       #include <iostream.h>
2
```

```
3      main()
4      {
5      float m=10;long d=50;
6      int j(m);
7
8      cout << "The Value of j is: " <<j <<endl;
9
10     int dd(d);
11     cout << "The Value of dd is: " <<dd <<endl;
12     return 0;
13     }
```

قد تتساءل عن الفائدة المرجوة من هذا الكود ، في هذا الكود حاولنا تسليط الضوء على مميزات أنماط البيانات العادية وقد اخترنا int **وكما تلاحظ فلقد هيئنا قيمة المتغير** j **بقيمة المتغير** m **والذي هو من النوع** float **ثم في السطر العاشر قمنا بتهيئة المتغير** dd **(من النمط** int**) بقيمة المتغير** d **والذي هو من النمط** long **؛ إذا نظرنا للنمط** int **على أنه صنف فأنت تعرف أنه استدعينا في السطرين العاشر والسادس تابع البناء لهذا الصنف ، في المرة الأولى قمنا بتهيئته بوسيط من النمط** float **والأخرى من النمط** long **فكيف نفعل ذلك؟**
الإجابة بسيطة جداً وهي أننا قمنا بزيادة تحميل تابع البناء للنمط int **لتصبح تستطيع إستقبال أي نمط غير نمط** int **؛ لو لم نقوم بزيادة تحميل تابع البناء لكننا أنشئنا تابع أخرى مثلاً** getFloat( ); **ليستطيع الصنف التعامل مع البيانات من النوع** float **؛ أي فإنه كان من الممكن أن نبدل السطر السادس والسابع بما يلي:**

CODE

```
6      int j;
7      j.getFloat(m);
```

لاحظ كيف أصبح التعامل مع البيانات من نوع int **وتخيل أننا أثناء دراستنا للبرمجة سنقوم بفعل كل ذلك. أي أننا سنحفظ أسماء جميع الدوال الخاصة بجميع أنماط البيانات. من هنا تأتي فائدة التحميل الزائد لدوال البناء. لا بد عليك من أن تركز حينما تصنع صنفاً جديداً على أنه يؤدي مهمة واحدة وأن تبتعد عن التعقيد مهما أمكنك ذلك. سواء التعقيد الكودي الذي أنت تقوم به أو التعقيد على صعيد مستخدم الصنف. أي أنه لا بد أن تجعل الصنف الذي تصنعه بسيطاً ومنظماً ومرتباً وسهلاً لأي مستخدم يريد إستخدامه بدلاً من أن تجعل مستخدم الصنف ينسى برنامجه ويركز على فهم طلاسم كيفية إستخدام صنفك الخارق.**
سنقوم الآن بكتابة مثال جديد وسنركز على تطويره حتى مرحلة معينة ثم نتوقف لنترك لك المجال لتطويره بنفسك . وسنطلق على هذا الصنف num .
وفي أول تطوير لهذا الصنف سنجعله يقبل التهيئة من قبل النمط int **و** float **و** long.

CODE

```
class num
{
double itsNum;
public:
num(int x){itsNum=x;}
num(float x){itsNum=x;}
num (long x){itsNum=x;}
GetItsNum() const { return itsNum;}
};
```

وهذا هـو التابع main() لإختبار الصنف:

CODE

```
void main()
 {
 int i=12;float g=13;long k=15;
 num first(i),second(g),third(k);
 cout << first.GetItsNum() << endl  ;
 cout << second.GetItsNum()   << endl ;
 cout << third.GetItsNum()  ;
 }
```

وبإمكـانـنا إختبارها بهذه الطريقة الأكثر عـملية:

CODE

```
void main()
 {
 int i=12;float g=13;long k=15;
 num first=i,second=g,third=k;
 cout << first.GetItsNum() << endl  ;
cout << second.GetItsNum()   << endl ;
 cout << third.GetItsNum()  ;
 }
```

كما تـرى ففـي الإختبـار الأول قمنـا بتهيئـة العــناصر فقـط ؛ أمـا فـي الإختبـار الثـانـي فلقـد قمنـا بإسـناد القيـم مباشـرة دون وضـع القوسـين وجميعهـا صحيحة لكن الطريقة الثانية أفضل وأسهل وأيسـر وأكثـر عـملانـية وهـذا ما يجب عليك محاولة فعله طـول حياتك البرمجية.

إلا أنه لا يجب عليك الظن بأن هذه هـي الطريقـة وأنـه بإمكـانـك التعامـل مـع الصنف Num على أنـه نمط أفضل من الأنماط الأساسـية.بقـي الكثيـر الكثير من المواضيع التي لا بد أن نتكلم فيها وعـنها فاصبر.

حتى نفهم ما كتب سابقاً أو حتى نتقن ما تم شرحه فلا بد علينا من التعامل مع أصناف تتعامل مع مؤشرات:

CODE
```
class num
{
double *itsNum;
public:
num(int x){itsNum=new double;*itsNum=x;}
num(float x){itsNum=new double;*itsNum=x;}
num (long x){itsNum=new double;*itsNum=x;}
~num () { delete itsNum;}
GetItsNum() const { return *itsNum;}
};
```

وبإمكانك إختبار الصنف السابق ولكن هذه المرة سنتعمد أن نظهر لك خطأ وهو خطأ خطير بالطبع

CODE
```
1     void main()
2     {
3     int i=12;
4     num first=i,second=first;
5     cout << first.GetItsNum() << endl  ;
6     cout << second.GetItsNum()   << endl ;
7
8     cout << first.itsNum << endl;
9     cout << second.itsNum;
10    }
```

قمنا بتهيئة الكائن الثاني second بإسناد الكائن first إليه ؛ وبعد ذلك في السطرين الثامن والتاسع طبعنا عنوان المؤشر itsNum (حتى تنجح في ترجمة هذا المثال قم بتغيير itsNum من عضو خاص إلى عضو عام) الموجود في الكائن first والموجود في الكائن second وستجد أن لهما نفس العنوان ؛ وهذا أقل ما يطلق عليه أنه خطأ شنيع. فأي تغيير الآن في حالة الكائن second أو first سيتبعه تغيير في الكائن الآخر ؛ وهذا ما سنحاول حله في الفقرة التالية.

** تابع بناء النسخة:

قاعدة:
كل دالة تابعة لصنف ما ؛ لها نفس اسم الصنف فإنها تسمى دالة بناء.

القاعدة السابقة صحيحة ؛ إلا أن بعض هذه الدوال تعتبر حالات خاصة وضرورية ولن يعمل الصنف الذي تقوم بإنشاءه إلا بها ومن ضمن هذه الدوال تابع بناء النسخة وهي إحدى الدوال التي يزودك بها المترجم في حال عدم تزويد المترجم بها.
يجب أن تعلم أن تابع بناء النسخة هو تابع بناء طبيعي مثله مثل أي تابع بناء إلا أنه في هذه المرة تستخدم عندما تتعامل أنت مع كائنين اثنين (أو عدة كائنات) من نفس الصنف.

لنفرض أن لدي الصنف Test وقد أنشئت منه كائنين هما Test1 و Test2 وقد قمت بكتابة السطر التالي:
Test2(Test1);
الذي سيقوم به المترجم هو البحث عن دالة بناء ليقوم ببناء الدالة Test2 من خلاله وكما قلت سابقاً فهي ستكون في هذه الحالة دالة بناء النسخة الإفتراضي التي يزودك بها المترجم ولن يحدث أية مشاكل حتى وإن لم تكن قمت بتعريف تابع بناء النسخة. لكن لنفرض أن الصنف Test يحتوي على عضو مؤشر اسمه N*. الذي سيقوم به المترجم حينما يستدعي تابع بناء النسخة أنه سينسخ جميع الدوال والمتغيرات الأعضاء الخاصة والعامة من الكائن (Test1) إلى الكائن (Test2) وبالتالي فإن المؤشر N* التابع للكائن Test1 سيكون هو نفسه التابع للكائن Test2 لأنهما يشيران إلى نفس منطقة الذاكرة.
والحل الوحيد الممكن هو أن تقوم بإنشاء دالة بناء جديدة تقوم بحجز الذاكرة للمؤشر N حتى لا يشير إلى نفس المكان.

تعريف: **دالة بناء النسخة: هي دالة بناء عادية إلا أن الوسيط الذي يمرر إليها هي إشارة إلى كائن من نفس الصنف.**

الآن دعنا نعد إلى المثال السابق والذي عرضنا فيه هذه المشكلة وسنقوم بكتابة دالة بناء النسخة حالاً وأرجو أن تكون يسيرة الفهم لك.

CODE

```
1    num::num(const num &rhs)
2    {
3     itsNum=new double;
4    *itsNum=rhs.GetItsNum();
5    }
```

لا تنسى أن تكتب تصريح دالة بناء النسخة في القسم العام من الصنف حتى تتم ترجمتها.
في السطر الأول قمنا بتمرير إشارة الكائن الممرر وهو كائن من نفس الصنف ولكن هذه المرة بإشارة ثابتة. ولقد أطلقنا على الصنف اسم rhs وهو من تقاليد التسمية المتبعة طبعاً . في السطر الثالث حجزنا للمؤشر itsNum ذاكرة. وفي السطر الرابع قمنا بتهيئة المؤشر بالقيمة التي يشير إليها المؤشر itsNum الخاص بالكائن الممرر.
أعلم أنك لم تفهم جيداً ولكن دعني أعيد كتابة المثال بأكمله ثم نشرحه جيداً.

CODE

```
1    #include <iostream.h>
```

```
2
3     class num
4     {
5     public:
6     double *itsNum;
7     public:
8     num(int x){itsNum=new double;*itsNum=x;}
9     num(float x){itsNum=new double;*itsNum=x;}
10    num (long x){itsNum=new double;*itsNum=x;}
11    ~num () { delete itsNum;}
12    num(const num &rhs);
13    GetItsNum() const { return *itsNum;}
14    };
15    num::num(const num &rhs)
16    {
17     itsNum=new double;
18    *itsNum=rhs.GetItsNum();
19    }
20    void main()
21     {
22     int i=12;
23     num first=i,
24    second=first;
25     cout << first.GetItsNum() << endl  ;
26     cout << second.GetItsNum()   << endl ;
27
28     cout << first.itsNum << endl;
29     cout << second.itsNum;
30     }
```

بالنسبة للسطر للخامس فهو لجعل المؤشر itsNum متغيراً عاماً حتى نتمكن من طباعة عنوان الذاكرة الذي يشير إليه في السطرين 28و29. حينما يتم تنفيذ البرنامج فلن تحدث أية مشاكل حتى نصل للسطر 24. والذي هو:

```
24    second=first;
```

هذا السطر لن تتم ترجمته هكذا بل بالأصح سيترجم هكذا:

```
24    second(first);
```

وحينما يصل المترجم إلى هذا السطر يبدأ في البحث عن الدالة وهي كما تلاحظ دالة خاصة بالكائن second وسيجدها في السطر 15 وسيمرر لها

الكائن first. وستقوم الدالة بإستقبال عنوان الكائن first وليس الكائن نفسه. في السطر 17 سيحجز المترجم للمؤشر itsAge ذاكرة جديدة كلياً (وبالتالي لن يشير إلى نفس المنطقة مع المؤشر itsAge الخاص بالكائن first) ، تخلصنا الآن من مشكلة أن المؤشران يشيران إلى نفس الذاكرة ولكن ظهرت مشكلة جديدة وهي أنه لا قيمة للمؤشر الجديد itsAge . إلا أن السطر 18 يحل هذه المشكلة نهائياً وهو:

```
18    *itsNum=rhs.GetItsNum();
```

وحتى نفهم ما يعنيه السطر 18 فربما نبسطه بالشكل التالي:

```
18    *itsNum=first.GetItsNum();
```

سيقوم هذا السطر بتهيئة المؤشر الجديد بالمؤشر itsAge الخاص بالكائن first من خلال دالة الوصول.

حينما يستمر تنفيذ البرنامج ستجد أن الرقمان الذان يطبعهما البرنامج في السطر 28 و 29 مختلفان كلياً.

الخطوة القادمة:

بالرغم من أننا طورنا الصنف num ليصبح بإمكانه التعامل مع المؤشرات والتعامل أيضاً مع الأنواع int و float و long ، إلا أننا بعد لم ننتهي وفي الحقيقة ما زلنا في البداية. بالنسبة للخطوة القادمة فهي تأتي لحل مشكلة بسيطة جداً وهي كالتالي:

```
24    second++;
```

حيث أن second هو كائن من الصنف num.
بالرغم من بديهية السطر السابق إلا أنه لن تتم ترجمته والسبب في ذلك يعود في أن المترجم لن يدري ماذا تعني (++) بالنسبة للصنف num (صحيح أنه يعلم ماذا تعني في الأنماط الأخرى) لأنها (أي العملية ++) غير معرفة بالنسبة للصنف num. لذلك فيجب عليك أن تقوم بتعريف ماذا تعني هذه العملية ++ حتى يفهم المترجم ماذا تقصد ويجب عليك أن تضمنها طرق للتعامل مع أنماط مختلفة غير نمط الصنف حتى يكون الصنف الذي تقوم بإنشاءه نمطاً يشار إليه بالبنان.
باختصار الخطوة القادمة هي التحميل الزائد للمعاملات.

كتابة أول معامل للصنف num:

سنحاول في هذه الفقرة محاولة تعريف (++) للصنف num . وحتى نضمن سهولة المادة العلمية المقدمة ؛ فسنبدأ أول بما هو بديهي وبما يجب عليك أن تفكر فيه أنت ، وهو أن تقوم بإضافة دالة جديدة في القسم العام للصنف تسمى Increament( ) أما عن تعريف هذه الدالة فهو:

```
1      num::Increament()
2       { return *itsNum++ ; }
```

وحتى نقوم بزيادة الصنف فإنه يجب علينا القيام بهذا:

```
1         num first=4;
2         first.increament();
```

هذه الطريقة غير عملية بتاتاً ، بالرغم من أنها صالحة .. فما أجمل من أن تكتب:

```
2         First++;
```

تـزودك الســي بلـس بلـس بإمكــانية فعـل هـذه الطريقــة ؛ كمـا تعلـم فـإن المعاملات تقسم إلى نـوعيـن:

1- معاملات أحاديـة:مثل ++ و - - .

2- معاملات ثنائيـة: مثل + و – و * و / .

والذي الآن سنقوم بمحاولة فعله هـو معامل أحـادي وهـو ++ .

> زيادة تحـميل المعاملات الأحاديـة:
> دالة بناء النسخـة: هـي دالة بناء عاديـة إلا أن الوسيط الذي يمرر إليها هـي إشارة إلى كائن من نفس الصنف.

حسب القاعـدة السابقـة فإنـه بإمكــانك بالفعـل تطـوير الصـنف num ليصـبح قابلاً للزيادة. ولكن هذا التطوير الـذي سـنقوم بــه سـيفتح لنــا أبـواب أخــرى للتطوير وهذا هـو الكـود بعد تطويره.

CODE

```
1      #include <iostream.h>
2
3      class num
4      {
5
6      double *itsNum;
7      public:
8      num() {itsNum=new double ;itsNum=0;}
9      num(int x){itsNum=new double;*itsNum=x;}
10     num(float x){itsNum=new double;*itsNum=x;}
11     num (long x){itsNum=new double;*itsNum=x;}
12     ~num () { delete itsNum;}
13     void setItsNum(double x) {itsNum=&x;}
14     num(const num &rhs);
15     GetItsNum() const { return *itsNum;}
16     num operator ++ ();
17     };
18
19     num::num(const num &rhs)
20     {
21      itsNum=new double;
22     *itsNum=rhs.GetItsNum();
23     }
24     num num::operator ++ ()
25     {
26     ++(*itsNum);
27     double x=*itsNum;
```

```
28      num temp;
29      temp.setItsNum(x);
30      return temp;
31      }
32      void main()
33       {
34       int i=12;
35       num first=i;
36       ++first;
37       cout << first.GetItsNum() << endl  ;
38       num second= ++first;
39       cout << second.GetItsNum() << endl  ;
40       }
```

بالرغم من صعوبة المثال السابق (للمبتدأين) إلا أنه يعد قفزة نوعية للأفضل إذا فهمته فهماً جيداً.
تغير الصنف num كثيراً . فكما ترى قمنا بإضافة ثلاث دوال.وهي كما يلي:
الدالة الأولى: (num) وهي كما ترى دالة بناء ، هذه الدالة تمنحك الكثير فالآن أصبح بإمكانك ، كتابة هذا السطر دون أن يعطيك المترجم أية أخطاء:

```
39      num first;
```

الدالة الثانية: (setItsNum) كان من المفترض أن توضع هذه الدالة سابقاً (أثناء البدايات الأولى للصنف) إلا أننا لم نتذكر فائدة هذه الدالة إلا حينما احتجناها (سترى في ماذا) . حينما تعمل على أي صنف. لا تنسى أن تضع محددات الوصول (دوال الوصول) لكل عضو متغير في الصنف.

الدالة الثالثة: (++ operator) هذه الدالة هي التي ذكرتنا بالدالتين السابقتين وفائدتهما. هذه الدالة هي التي تجعل من السطرين 36 و 38 صحيحة.

```
1       num num::operator ++ ()
2       {
3       ++(*itsNum);
4       double x=*itsNum;
5       num temp;
6       temp.setItsNum(x);
7       return temp;
8       }
```

أول ما يجب عليك ملاحظته أن لهذه الدالة قيمة إعادة (لا تنسى هذا الأمر) وهي من نفس نوع الصنف. في السطر الثالث قمنا بزيادة المتغير الرئيسي في الصنف حتى قد تقول أنه الآن كل شيء انتهى (تستطيع التوقف الآن ولكن بشرط أن تغير تصريح الدالة السابقة ليصبح هكذا: void ( ) operato++r) ولكن إذا توقفت الآن فإن السطر 38 لن تتم ترجمته . بالنسبة للسطر الرابع فلم أضعه إلا خوفاً من خطورة المؤشرات وحتى

أضمن عدم خطورتها فلقد قمت بإسناد قيمة المؤشر itsNum (التابع إلى الصنف) إلى متغير جديد وهو x ثم نرسل المتغير x إلى الدالة setItsNum الخاصة بالكائن الجديد temp (الذي أنشأناه في السطر الخامس) .
ينتقل الآن تنفيذ البرنامج إلى الدالة setItsNum والتي لا وظيفة لها إلا أنها تقوم بإسناد القيمة الممررة إليها إلى المتغير الرئيسي فيها وهو itsNum (الخاص بالكائن temp) . الآن تعود الدالة ++ operator بالكائن temp وينتهي تنفيذها.

بعد أن شرحنا تنفيذ هذه الدالة. فكل ما علينا فهمه الآن هي كيفية عملها أثناء تنفيذ البرنامج.
ودعنا الآن ننتقل إلى الدالة main لنحاول فهم البرنامج من خلالها.

```
1     void main()
2      {
3      int i=12;
4      num first=i;
5      ++first;
6      cout << first.GetItsNum() << endl  ;
7      num second= ++first;
8     cout << second.GetItsNum() << endl  ;
9      }
```

لا إشكالية في الأسطر الأربع الأولى . ولكن يبدأ تنفيذ الدالة ++ operator في السطر الخامس ضمن الكائن first . انتقل الآن إلى الدالة ++ operator وستقوم بما يتوجب عليها فعله. بالنسبة للسطر السابع فحاول أن تفهم الآن كيف يترجمه المترجم:

```
7      num second(++first);
```

وبتحديد أوضح سيكون السطر المترجم كالتالي:

```
7      num second= (first.operator++( ) );
```

أي أن المترجم سيقوم بتنفيذ الدالة ++ operator الخاصة بالكائن first أولاً والتي تعود بالكائن الجديد temp ليمرر إلى دالة بناء النسخة الخاصة بالكائن second ثم تنفذ دالة بناء النسخة دون أية مشاكل.
لقد نجحنا في تنفيذ تطويرات كثيرة وكبيرة بالنسبة للصنف num .
إلا أنه بالرغم من هذا فهناك بعض العيوب والتي كان من الممكن تلافيها.
فبالنسبة لتعريف المعامل ++ فإنك تقوم بإنشاء كائن جديد مؤقت. وهذا بدوره سيؤثر على السرعة والوقت والذاكرة بالنسبة للجهاز، وخاصة إذا احتوى برنامج على آلاف الأسطر.
الذي نريده الآن هو ضمان السرعة والذاكرة التي تذهبان سدىً دون أي فائدة.

فائدة للمؤشر this:

كما تعلمت سابقاً فإن المؤشر this يشير إلى الكائن الذي يحتويه. إذا قمنا بإنشاء إشارة أو مرجعية لهذا الكائن فإن الدالة ستعيد الكائن نفسه. إذاً يصبح بإمكاننا تغيير الدالة ( ) ++ operator لتصبح هكذا بعد التعديل:

```
1     num num::operator ++ ()
```

```
2	{
3	++(*itsNum);
4	return *this;
5	}
```

فإن الصنف num سيصبح أكثر تميزاً وأكثر سهولة وسلاسة.
بنفس الطريقة السابقة التي شرحناه بإمكانك زيادة تحميل المعامل (- -).

المعامل اللاحق:

الصنف num لا يعالج سوى المعامل السابق، ولا يستطيع معالجة المعامل اللاحق فلو عدلت السطر 38 في البرنامج السابق هكذا:

```
38	num second= first++;
```

فإن المترجم سيصدر تحذير فقط. ولكن عند التنفيذ ستجد عدة أخطاء كبيرة جداً ، فالبرنامج الآن سيزيد الكائن first ثم يسند القيمة إلى second بالرغم من أن المطلوب إسناد first إلى second ثم زيادة الكائن first .
لحل هذه المشكلة فأحد الإقتراحات هو زيادة تحميل الدالة operator ++( ) لنستطيع تمرير متغير إليها ، هذا المتغير سيكون المؤشر itsAge الخاص بالكائن first ، ثم تقوم الدالة بإنشاء كائن جديد مؤقت توضع فيه المؤشر itsAge ثم تزاد قيمة الكائن first وتعود الدالة بالكائن المؤقت وتسنده إلى الكائن second. أي تعريف الدالة الجديدة هو كالتالي:

```
1	num::num operator ++ (int  m)
2	{
3	num temp(*this);
4	++itsNum;
5	return temp;
6	}
```

وبالرغم من صحة المثال السابق إلا أنه بسبب أننا جعلنا المتغير itsNum مؤشراً وليس متغيراً عادياً . فإن كل هذا جعل عمل الصنف يتغير كلياً عندما نحاول تعريف المعامل اللاحق.

بالرغم من كثرة الأكواد وشروحها في هذه الوحدة إلا أنها ستزيد من قدراتك البرمجية كثيراً ، فحاول فهمها.

هذا هو الكود بحلته الجديدة:

CODE

```
1	#include <iostream.h>
2	
3	class num
4	{
5	double *itsNum;
6	public:
```

```
7     num() {itsNum=new double ;itsNum=0;}
8     num(int x){itsNum=new double;*itsNum=x;}
9     num(float x){itsNum=new double;*itsNum=x;}
10    num (long x){itsNum=new double;*itsNum=x;}
11
12    num (double x){itsNum=new double;*itsNum=x;}
13    ~num () { delete itsNum;}
14    void setItsNum(double x) {itsNum=&x;}
15    num(const num &rhs);
16    double GetItsNum() const { return *itsNum;}
17    num operator ++ ();
18    num operator ++ (int m);
19    };
20
21    num::num(const num &rhs)
22    {
23     itsNum=new double;
24    *itsNum=rhs.GetItsNum();
25    }
26     num num::operator ++ ()
27    {
28    ++(*itsNum);
29    return *this;
30
31    }
32     num num::operator++ (int m)
33    {
34
35          return num((*itsNum)++);
36    }
37
38    void main()
39     {
40     int i=12;
41     num first=i;
42     first++;
43     cout << "first++ :\t" << first.GetItsNum() << endl  ;
44     num second= first++;
```

```
45      cout << "first++       " << first.GetItsNum() << endl  ;
46      cout << "second  \t" << second.GetItsNum() << endl  ;
47    }
```

بالنسبة لتعريف المعامل السابق فلقد عدنا مرة أخرى إلى التعامل مع المؤشر this

أما بالنسبة لتعريف المعامل اللاحق فهو يبدأ من السطر 32 إلى السطر 36. وهو لا يحتوي إلا على سطر واحد هو :

```
35            return num((*itsNum)++);
```

وكما ترى فأنت لا تعلم مالذي تعيده الدالة ، تعلم أن نوع القيمة المعادة هي الصنف num ، السبب في عدم وجود كائن هو أن السي بلس بلس تسمح لك بفعل ذلك فأنت بإمكانك أن تكتب في نهاية الدالة main( ) هذا السطر:

```
            return int (0)
```

وهذا بالطبع ما يمكنك فعله ، أي باختصار تستطيع إعادة كائن غير مسمى؛ بالنسبة للسطر 35 فإن المترجم سيترجمه هكذا:

```
1            num temp (*itsNum);
2            ++(*itsNum);
3            return temp;
```

في السطر الأول ولأغراض الشرح فلقد أطلقنا على الكائن المعاد اسم temp (في السطر الأصلي الكائن المعاد ليس له مسمى) نقوم بتمرير المؤشر itsNum إلى دالة البناء الخاصة بالكائن temp وكما ترى فهنا ستظهر فائدة السطر الجديد رقم 12 حيث يقوم بإستقبال متغير من النوع double أي أننا حالياً قمنا بنسخ الكائن الأساسي إلى الكائن المؤقت temp . في السطر الثاني قمنا بزيادة المتغير itsNum الخاص بالكائن الأساسي. وفي السطر الثالث أعدنا الكائن المؤقت. وحتى تفهم جيداً فحاول تطبيق الشرح السابق على هذا السطر:

```
44     num second= first++;
```

إلى هنا لقد انتهينا من شرح المعامل اللاحق. وأملي أن تحاول فهم هذا الكلام المشروح حتى يصبح ذا فائدة على الأقل.

إذا فهمت ما سبق. فحاول زيادة تحميل المعامل (- -) واجعله يدعم الطريقتين السابق واللاحق.

المعاملات الثنائية:

معاملا الزيادة والنقصان هي من المعاملات الأحادية وبالتالي فهو يعمل على كائن واحد فقط. أما المعاملات الثنائية فهي تعمل على كائنين سنحاول الآن تطوير الصنف num ليصبح قادراً على فعل الآتي:

```
1            num One,Two=2,Three=3
2            One= Two+Three;
```

وهكذا بالنسبة للمعاملات الأخرى مثل الضرب والطرح والقسمة وغيرها.

المعامل (+):

سنطوّر الآن الصنف num ليشتمل على القدرة مع التعامل مع المعامل + . والدالة التي سنقوم بزيادة تحميلها هي دالة + operator وإليط طريقة تعريف هذا المعامل الجديد:

```
1     num num::operator+ (const num &rhs)
2     {
3     return num (*itsNum+rhs.GetItsNum());
4     }
```

بالنسبة للسطر في الدالة main والذي يستخدم هذه الدالة فهو كما يلي:

```
1           num One,Two=2,Three=3
2           One= Two+Three;
```

في السطر الأول صرحنا عن ثلاث كائنات تنتمي للصنف num وقد هيئنا كل كائن بقيمة أما الثالث فأسندنا له مجموع قيمة الكائنين السابقين. السطر الثاني يترجم هكذا:

```
2           One=Two.operator+(const num &three);
```

لقد تعرض الصنف num لمشاكل كبيرة وذلك نظراً لتعامله مع المؤشرات ، من أجل ذلك وبالرغم من حرصي على عدم أن يحمل هذا الصنف أي خطأ مهما كان نوعه، فلقد حمل بالفعل خطأ لم أعلم به إلا حينما طبقت زيادة تحميل المعامل + . وقد كانت عواقبه خطيرة للغاية. حيث أنه يوقف نظام التشغيل بالكامل. وهذا الخطأ بسيط جداً وهو أننا في إحدى المراحل حينما نقوم بنسخ كائن لكائن آخر (وخاصة في المعاملات اللاحق و +) فإن الكائن الأول يهدم وحسب دالة الهدم فإن المؤشر سيلغى وبالتالي يصبح هناك مؤشر هائم مما يؤدي إلى كارثة. فإذا طبقت هذا الخطأ على الفيجوال سي فستظهر أرقام غريبة للغاية ولن يكشف لك الخطأ. أما بالنسب لبورلاند سي بلس بلس فلن يعترض وسيقوم بالواجب ولكن إذا أغلقت نافذة تنفيذ البرنامج فقد يتوقف نظام التغشيل بالكامل. هناك حلان لهذه المشكلة وهي إما أن نقوم بإعادة كتابة الصنف num ليصبح itsNum متغير عادي أو نقوم بتعديل دالة الهدم لتصبح هكذا:

```
13    ~num () {  }
```

بالنسبة لي أنا فإني أفضل القيام بالحل الأول. فما هو الداعي لإستخدام المؤشرات وكما تعلم فهي ميزة خطيرة للغاية، سأعيد كتابة الصنف num من جديد ولكن هذه المرة بجعل المؤشر itsNum متغيراً عادياً وسأترك لك مهمة تطويره؛ هناك حل ثالث وهو الأفضل من بينها جميعاً ، سنصل إليه حالاً، وحتى نضمن قدرتك على تطوير صنف يحوي مؤشرات وصنف آخر لا يحويها فسأعيد كتابة الصنف num لكن هذه المرة بدون مؤشرات

```
1. class num
2.       {
3.
4.       int itsNum;
5.       public:
6.       num() {itsNum=0;}
7.       num(int x){itsNum=x;}
8.       num(float x){itsNum=x;}
```

```
9.       num (long x){itsNum=x;}
10.            num (double x){itsNum=x;}
11.            ~num () { }
12.            void setItsNum(int x) {itsNum=x;}
13.            num(const num &rhs);
14.            double GetItsNum() const { return itsNum;}
15.            const num &operator ++ ();
16.            const num operator ++ (int m);
17.            num operator+ (const num &rhs);
18.
19.            };
20.
21.            num num::operator+ (const num &rhs)
22.            {
23.            return num  ((itsNum)+rhs.GetItsNum());
24.            }
25.
26.
27.            num::num(const num &rhs)
28.            {
29.
30.            itsNum=rhs.GetItsNum();
31.            }
32.            const num& num::operator ++ ()
33.            {
34.            ++itsNum;
35.
36.
37.            return *this;
38.            }
39.            const num num::operator++ (int m)
40.            {
41.            num temp(*this);
42.            ++itsNum;
43.            return temp;
44.            }
```

بعد أن انتهينا من شرح معامل الجمع (+) فأعتقد أنه أصبح بإمكانك زيادة تحميل بقية المعاملات الثنائية مثل الضرب والطرح والقسمة.

ولكن تظل هناك مشكلة أخرى وهي حينما نقوم بكتابة السطر التالي:

```
1        second=i+first;
```

حيث i هو متغير من النوع int فسيصدر المترجم خطأ.
لكي تحل هذه المشكلة فكل ما عليك هو تعريف دالة معامل + غير عضو في الصنف num ، ونظراً لأنها غير عضو فلن تصل إلى المتغير itsNum لأنه خاص ولحل هذه المشكلة فيجب عليك تعريفه بأنه صديق للصنف num وهذا سيحل المشكلة:

```
1    num operator+ (double x,const num &rhs)
2    {
3    return num (x+rhs.GetItsNum());
4    }
```

تذكر: أن تقوم بالتصريح عن هذه الدالة كدالة صديقة في جسم تصريح الصنف num.

بعد أن انتهينا من تعريف المعامل + ، فلا بد الآن أن نخرج قليلاً عن زيادة تحميل المعاملات على أمل الرجوع إليها مرة أخرى بعد أن ننتهي من مواضيع أخرى متصلة بزيادة التحميل.وقبل ذلك فلا بد أن ننتهي من تعريف عملية مهمة جداً ألا وهي عملية الإسناد (=).

زيادة تحميل المعامل (=):

الدالة الأخيرة التي يزودك بها المترجم إفتراضياً هي دالة المعامل (=).
يأتي هذا المعامل لحل المشاكل المتعلقة بالإسناد والتي لا يستطيع حلها دالة بناء النسخة. وبالرغم من أن نسخة الصنف num التي تحوي مؤشرات قد وصلت إلى فشل ذريع بسبب وجود المؤشرات الهائمة فإن زيادة تحميل المعامل = ، هي الضمانة الوحيدة لعمل الصنف num دون أن يشتكي من أية مشاكل ، سنترك لك هذه المهمة حتى تحلها بنفسك ، سنعطيك فكرة الحل ونطبقها على مثال آخر، إذا افترضنا بأن لدينا كائن من الصنف num وأنك قمت بإسناده إلى نفسه، فإن الذي سيحدث حقيقة هو أن الصنف الذي على الجانب الأيمن من عملية الإسناد سيتهدم وتتهدم معه مؤشراته أو تصبح مؤشرات هائمة وحينما يصبح الكائن جاهزاً للنسخ إلى الكائن الذي على الجانب الأيسر (وهو نفسه)، فإن ذلك يعني أنك دمرت الكائن ولم تسند أي شيء أو بالمعنى الأصح قمت بإسناد كائن مهدم إلى كائن على وشك البناء . سنأخذ الآن مثال وسنحاول فهم مالذي يحدث بالضبط؟.

CODE

```
1. #include <iostream.h>
2.
3. class Bird
4. {
5.       int *itsAge;
6. public:
```

```
7.        Bird () {itsAge=new int(2);}
8.        GetitsAge()const { return *itsAge;}
9.        SetitsAge(int age) {*itsAge=age;}
10.
11.             Bird & operator = (const Bird &rhs)
12.             {
13.                   if (&rhs == this) return *this;
14.                   *itsAge=rhs.GetitsAge();return *this;
15.             }
16.       };
17.
18.       void main()
19.       {
20.             Bird a,b;
21.             a.SetitsAge(6);
22.             b=a;
23.             cout << "b=   " << b.GetitsAge();
24.       }
```

هـذا الكــود لا يصـنع أي شــيء. وإنمـا مـا وضـع إلا لشــرح مفهــوم عــمل المعامل(=) كما تلاحظ حسب رأس الدالة

```
25.       Bird & operator = (const Bird &rhs)
```

فإنها تعـيد قيمـة مرجعية من النـوع bird أما عـن وسائطها فهـي إشارة إلـى كائن ممرر وكما ترى في دالة main( ) :

```
26.       b=a;
```

فإنه سيترجم هـكذا:

```
27.       b.operator =(a);
```

أي أنـك حينمـا تقـوم بإسـناد الكـائن a إلـى الكـائن b فإنـك فـي الحقيقـة تستدعـي الإجراء operator وتقوم بتمريـر الكـائن a إليـه ثـم يسـتمر التنفيـذ على هذا الشكل:

```
28.       {
29.                   if (&a == this) return *this;
30.                   *itsAge=a.GetitsAge();return *this;
31.             } //  قمنا بتعديل أسماء الكائنات للفهم
```

في السطر 29 يتأكد أنك لا تقوم بإسناد الكائن إلـى نفسـه وفـي حـال قمـت فإنـه يعـود بإشارة إلى الكائن b (لاحـظ المؤشـر this يشـير إلـى الكـائـن b) . أما في حال أنك لم تقم بفعل ذلـك فـإن التنفيـذ ينتقـل إلـى السـطر الثلاثـين حيـث يقـوم بإسـناد المؤشـر itsAge الخـاص بالكـائن b إلـى المؤشـر itsAge

الخاص بالكائن a ، لاحظ أنه يقوم بإسناد القيم وليس العناوين. بعد ذلك تعود الدالة بإشارة إلى الكائن b.
بعد أن انتهينا من موضوع زيادة تحمل المعامل (=) فلقد آن لنا أن ننتقل إلى مواضيع أخرى وتغيير هذا الموضوع ثم العودة إليه لاحقاً.

تحويل الأنماط:

حتى تعمل المعاملات السابقة فلا بد من إحتواءها على معاملات تحويل الأنماط فهي التي تمكننا من إسناد القيم ذات الأنماط المختلفة إلي بعضها.
لقد مررنا بالفعل على بعض معاملات التحويل وهي تلك التي تحوّل الأنواع الداخلية مثل int و float وغيرها إلى الصنف num . وهي كما ترى جميع دوال البناء:

```
45.            num() {itsNum=0;}
46.            num(int x){itsNum=x;}
47.            num(float x){itsNum=x;}
48.            num (long x){itsNum=x;}
```

وهذه هي طريقة تحويل الأنماط الداخلية إلى الصنف num. أما إذا احتوى الصنف على مؤشرات فخير طريقة لفهم معاملات التحويل هي الرجوع إلى الصنف num الذي يحتوي على معاملات التحويل وهي كما ترى:

```
7      num() {itsNum=new double ;itsNum=0;}
8      num(int x){itsNum=new double;*itsNum=x;}
9      num(float x){itsNum=new double;*itsNum=x;}
10     num (long x){itsNum=new double;*itsNum=x;}
11
12     num (double x){itsNum=new double;*itsNum=x;}
```

والفرق بينها وبين السابقة هي إحتواءها على عمليات حجز الذاكرة.

والآن كيف نستطيع التحويل من الصنف num إلى النمط int ، تقوم السي بلس بلس بتزويدك بالمعاملات المناسبة لفعل ذلك. وسنحاول في هذه الأسطر محاولة إسناد متغير من النوع int إلى الصنف num ، بالطبع لن أعيد كتابة الصنف num بل سأعيد ما هو مهم بالضرورة:

```
1. #include <iostream.h>
2.       class num
3.       {
4.       int itsNum;
5.       public:
6.       num() {itsNum=0;}
7.       num(int x){itsNum=x;}
8.       ~num () { }
9.       void setItsNum(int x) {itsNum=x;}
10.      num(const num &rhs);
11.      int GetItsNum() const { return itsNum;}
```

```
12.      operator int();
13.      };
14.            num::num(const num &rhs)
15.            {
16.            itsNum=rhs.GetItsNum();
17.            }
18.
**       num::operator int()
**       {
**       return (int (itsNum) );
**       }
19.
20.            void main()
21.             {
22.             int i=12;
23.             num first=i;
24.             int j=first;
25.             cout <<"j:   "<< j << endl  ;
26.             cout <<"first:   "<< first.GetItsNum() << endl  ;
27.             }
```

ما يهـم في هذا البرنامج هـو السطر 24 وهـو كالتالي:

```
28.             int j=first;
```

والذي سيترجم هـكذا:

```
29.             int j=first.operator int();
```

أي أن التنفيـذ سـينتقل إلـى الدالـة operator int( ) الخاصــة بالكـائن first والقيمة التي سيعـود بها ستسند إلى المتغير j . أي أننا سننتقل من السـطر 24 إلى هذا المقطع من البرنامج:

```
1        num::operator int()
2        {
3        return (int (itsNum) );
4        }
```

ما يهـمنا هـو السطر الثالث حيث تعيد الدالة متغيراً غيـر مسمــى مـن النــوع int وتقوم بتهيئتـه بالمتغير itsNum الخاص بالكائن first .

وبهذه الطريقـة يمكـنك إضـافة معـاملات تحــويل أخــرى مثـل: float و long و double .

أحـد أكبر وأهـم فوائد معاملات التحـويل هـي أنك تـتخلص مـن محاولـة زيـادة تحـميل المعاملات الثنائية لتصبح قادرة على التعامل مع أنماط أخرى.

لقد إنتهينا الآن بالفعل من هذا الموضوع (موضوع التحـــميل الزائـد) وخاصــة بعـد أن أنهينا موضوع زيادة تحـميل المعاملات الثنائيـة.

حتى تزيد من معرفتـك بزيـادة تحـــميل المعـاملات فبإمكــانك مراجعــة قســم الأمثلة التطبيقيـة حيث أننا قمنـا بزيـادة تحـــميل بعـض المعـاملات والتـي لـم نتحـدث عـنها هـنا.

عيوب التحـميل الزائد:

أحد أكبر عيوب التحـميل الزائد هـو محاولة الإستخـدام غير الشرعية وتعــديل بعض الوظائف الأساسية كجعل الجمـع يطرح بدلاً من أن يجمع.
هــذا الغـــموض بالرغــم مــن متعـتـــه ســيؤدي إلـــى إنشــاء أصـناف لا تصـلح للإستخــدام مما يؤدي إلى ضياع الوقت والجهـد.
حتـى تسـتفيد مـن زيـادة التحـــميل فبإمكــانك إستخـدامــه فـي إحـدى هـذه المجالات:

- إنشــاء أنــواع جـــديدة كليـاً كالأنمـاط الداخليـة لهـا ميـزات أعلـى منهـا كقدرتها على التعامل مع الأعـداد التخيلية (المركبـة).
- زيـادة تحــميل دوال البنـاء والـدوال الأخـــرى مما يجعـل مـن وظيفـــة مستخـدم الصنف سهلة للغاية وحتى تقلل مـن تعـليمات الإستخـــدام لصنفك.

بالرغـم من هذه الميزات إلا أن هـناك بعض العيوب:

- عيوب خاصـة في الوراثـة: فعـندما يرث صنف ما صنف آخر وقام بتجـاوز إحدى دالات الصنف الأب ، فإن الدوال المحمـلة الأخـرى تلغـى أيضاً.
- عـدم إمكـانية إنشاء معاملات جـديدة كمعامـل ** والـذي مـن الممكـن إستخـدامـه لإيجاد مربع عـدد ما.
- لـيس بإمكــانك زيـادة تحـــميل معامـل أحـادي للقيـام بوظيفـــة معامـل ثنائي.
- ليس بإمكـانك تغيير أسبقية المعاملات الحسابية.

هـناك بعض المعاملات التي لم نذكر كيفية زيادة تحـــمليها ، الأمثلـة القادمــة تحاول فعل ذلك.

زيادة تحميل المعامـل( ):

هل تتذكر المصفوفة الديناميكيـة والذي أتت به إلينا ، هـناك ما هـو أفضل من المصفوفة الديناميكية ألا وهـي المتجهـات ، سنتعرف إليها بشـكل عـام فـي آخر وحدة ، المتجهـات بإمكانك تحـديد حجمها فـي أي وقـت تشـاء ومتـى مـا أردت فلو اخترت أن تكون فـي البدايـة 100 عـــنصر ثم قـررت أن ترفعهـا إلـى 200 عـنصر ثم قررت أن تخفضها إلى عــنصر واحـد فلـن تعتـرض أبـداً بعـــكس المصفوفة الديناميكـية والسبب في ذلك ليس حجمها الكبير وإنما في قدرتها على تخصيص الـذاكرة وإلغـاء تخصيصـها بواسطــة المؤشـرات ، سـأترك لـك فرصـــة تطـوير المتجهـــات بنفســك ، أما الآن فسـنتعرف علـى كيفيـة تحـميل المعامل ( ) ، أنظر إلى هذا المثال:

CODE

```
1. #include <iostream>
2. using namespace std;
```

```
3.
4. class array
5. {
6.       int number;
7.       int *arrays;
8. public:
9.       array()
10.            {
11.                  int i=0,j=0;
12.                  number=100;
13.                  arrays=new int[number];
14.                  for(j=0, i=0;i<100;i++,j=10*i)
15.                  arrays[i]=j;
16.            }
17.            int operator() (int x)
18.            {
19.                  if(x>number) return 0;
20.                   else return arrays[x];
21.            }
22.
23.
24.      };
25.
26.      int main()
27.      {
28.            array a;
29.
30.            for(int i=0;i<10;i++)
31.                  cout << a(i) << endl;
32.
33.            return 0;
34.      }
35.
```

في السطر 4 تم الإعلان عن الصنف array ، هذا الصنف يتحكم في حجم مصفوفة ويدير عملياتها ، وفي هذا المثال بإمكانك تطويره ليصبح متجهاً.
في السطر 17 تم زيادة تحميل المعامل( ) ، حيث أن هذا المعامل يستقبل بارامتر واحد وهو رقم العنصر الذي تريد إعادته ، في حال كان الرقم الممرر أكبر من 100 أي زائد عن حجم المصفوفة فسيتم إعادة قيمة 0 أما إذا كان الرقم صحيحاً فسيتم إعادة العنصر الذي يريده المستخدم من المصفوفة.

مثال صنف الأعداد الكسرية Fraction

الهدف من المثال/

سنقوم في هذا المثال بكتابة صنف يتعامل مع الأعداد الكسرية وسنترك لك القدرة على فعل ما تريد فيه ، الغرض من هذا المثال هو إعطاءك نواة لفهم أفضل لكيفية إنشاء مثل هذا الصنف، ونحن لن نقوم بإنشاءه ليس لصعوبة المثال بل حتى نترك لك تمريناً تفهم من خلاله مواضيعاً مختلفة في السي بلس بلس.

الحل:

سنقوم بتصميم هذا الصنف كما يلي:

- **سنطلق على هذا الصنف اسم** Fraction **حتى يكون اسمه مماثلاً للغرض من الصنف.**
- **شكل الصنف هو هكذا** a/b **حيث** a **البسط و** b **هو المقام وهذه هي المتغيرات الأعضاء الخاصة ، وسنجعلها على نمط** int **حتى لا نسمح للمستخدم أن يضع أعداد عشرية مما يؤثر على الصنف بشكل كامل.**
- **هناك عضو متغير خاص جديد ألا وهو** num **من النوع** float **وهو الصيغة العشرية للعدد الكسري ونظراً لأن أي تعديل على هذا العدد سيجعل الصنف ينهار فلن نمكن المستخدم من تغييره على الإطلاق وسنغلفه ، ولن تقدر على التعديل عليه إلا بتغيير قيم الـ** a **و الـ** b.
- **نظراً لأن شكل الصنف هكذا** a/b **فسنقوم بتغيير طرق إدخال وإخراج الصنف، حيث سيكون بإمكان المستخدم إدخاله على صورته الطبيعية ، ونظراً لأن مستخدم الصنف سيحاول كتابة الصنف هكذا مثلاً:** a0b **، فلن نمكنه من فعل ذلك وسنجعل البرنامج ينتهي على الفور.**
- **سنقوم بتحميل المعاملات التالية:** + **و** * **و** / **، بالنسبة لعملية الجمع فلن تكون بين عددين من نفس الصنف ، فلقد تركنا هذه المهمة لك ، وبالنسبة لبقية المعاملات فلن نقوم بإعادة تحميلها وسنتركها لك:**

CODE

```
1. # include <iostream.h>

2. class Fraction
3. { /*               المتغيرات الأعضاء الخاصة          */
4. int up;
5. int down;
6. float num;
```

```
7. public:
8. /*        محددات الوصول        */
9. GetUp(){return up;}
10.   GetDown(){return down;}
11.   GetNum(){retur num;}
12.   SetUp(int x) {up=x;}
13.   SetDown(int x){down=x;}
14.   /*      دوال البناء           */
15.   Fraction():up(1),down(1),num(1){}
16.   Fraction(int a):up(a),down(1),num(a){}
17.   Fraction(int a,int b):up(a),down(b),num(a/b){}
18.   /*          تحميل المعاملات           /*
19.   Fraction operator+ (int rhs)
20.   {
        return Fraction (up+rhs,down);
21.   }
22.   Fraction operator* ( Fraction rhs)
23.   {
         return Fraction (up* rhs.GetUp() , down* rhs.GetDown()) ;
24.   }
          Fraction operator/ ( Fraction rhs)// 1/2     2/1
25.   {
        int m;
        m=rhs.GetUp();
        rhs.up=rhs.GetDown();
        rhs.down=m;
        Fraction temp(up* rhs.GetUp() , down* rhs.GetDown());
        return temp;
26.   }
27.   friend Fraction operator+ (int ,Fraction&);
28.   friend ostream &operator << (ostream& ,const Fraction &);
29.   friend istream &operator >> (istream& , Fraction &);
30.   };

31.   Fraction operator+ (int rhs,Fraction &temp)
32.   {
33.   return Fraction (rhs+temp.up, temp.down);
34.   }
```

```
35.   /*        دوال الإدخال والإخـراج        */

36.   istream& operator >> (istream& E, Fraction& temp)
37.   {
38.   E >> temp.up;
39.   char c;
40.   E.get(c);
41.   if (c !='/') throw;
42.   E >> temp.down;
43.   return E;
44.   }

45.   ostream &operator << (ostream& D ,const Fraction &temp)
46.   {
47.   return D << temp.up << "/" << temp.down ;
48.   }
```

هـناك بعض المواضيع التي قمنا بطرحهـا وسنبدأ بها واحداً واحداً:

نظرة عـامـة على الصنف:

لقد قمنا بتقسم الصنف إلى خمسـة أقسـام وهـي:
المتغيرات الأعضـاء الخاصـة: وهـي ثلاثـة متغيرات جـرى ذكرها فـي بدايـة شرح المثال.
محددات الوصول.
دوال البناء.
دوال تحـميل المعاملات.
دوال تحـميل معاملات الإدخال والإخراج.

محددات الوصول:

كما تعلم فإن هـناك دالتين نقوم بتحـديدها للوصول؛ هـما:
دالة set(int): وتستخـدم هذه الدالة لتغيير قـيم المتغيـرات المغلفـة داخل الصنف ، وبالنسبـة لخـطورة التعامل مع المتغير num) والذي هـو عبـارة عـن واجهـة العـدد العشري للصنف) فلقد قررنا ولحماية الصنف من أي تغيير عـدم السماح لأي كـان تغييره ومن أجل ذلك لم نضع له دالة SetNum(int) .
دالة Get( ) : تستخـدم هذه الدالـة للوصـول إلـي الأعضـاء المغلفـة داخل الصنف ، لأغراض المقارنـة أو الإسناد أو أي شـيء آخـر ، لكـن لـيس بإمكـانك إسنـاد إحدى القيم للدالة فهذا عبارة عـن خطأ ، ومـن أجـل عـدم حصـول أي خـطورة فلقد وضعـنا دالة GetNum( ) .

دوال البناء:

هـناك ثلاث دوال للبناء ؛ الاولـى لا تسـتقبل أي عـدد والثانيـة تسـتقبل عـدد واحد والثالثـة تستقبل عـددين .
بالنسبة للدالة الأولى فهـي تقـوم بتهيئة المتغيرات الأعضـاء بالقيمـة 1.

أما الدالة الثانية فهي تقوم بتهيئة البسط بالعدد الممرر إليها وتقوم بتهيئة المقام بالقيمة 1 وأيضاً تقوم بتهيئة العدد العشري بناتج قسمة البسط على المقام.

أما الدالة الثالثة فهي تقوم بتهيئة البسط بالعدد الأول الممرر إليها والمقام بالعدد الثاني الممرر إليها والعدد العشري بناتج قسمة البسط على المقام.

على مصمم أي صنف أن يجعل صنفه أكثر تماسكاً وفعالية وأن يكون سهل الإستخدام لذلك وضعنا في هذا الصنف بعض دوال البناء المحتمل وضعها وتركنا لك كيفية التفكير في بقية الإحتمالات ، ماذا لو أراد المستخدم تهيئة العدد الكسري بقيمة عشرية فماذا تفعل . كيف ستتصرف مع الجزء العشري من الرقم ، لذلك تركنا لك حل هذا الإحتمال وهو بسيط للغاية.

دوال تحميل المعاملات:

هذا القسم يبدأ من السطر 19 إلى السطر 28 وهو يقوم بتحميل عمليتين هما القسمة والضرب وجزء من عملية الجمع ، سنبدأ بشرحها واحدة واحدة.

```
Fraction operator* (Fraction  rhs);
```

تحميل هذا المعامل لا يكلف إلا سطر واحد وهو كالتالي:

```
return Fraction (up* rhs.GetUp() , down* rhs.GetDown()) ;
```

كما تعلم فإن عملية ضرب الأعداد الكسرية تعني ضرب بسط العدد الأول في بسط العدد الثاني ومقام العدد الأول في مقام العدد الثاني ، وهذا ما تقوم به دالة زيادة تحميل المعامل * ؛ وحتى نفهم السطر الوحيد الذي تتألف منه هذه الدالة دعنا نرى كيف يقوم المترجم بترجمتها:

```
Fraction temp;

Temp (up* rhs.GetUp() , down * rhs.Getdown);

Return temp;
```

دالة المعامل * تقوم بإعادة كائن مؤقت وهو كما ترى في السطر الاول قمنا بالإعلان عنه أما في السطر الثاني فلقد قمنا بإعادة بنائه من جديد برقمين اثنين ، الأول هو مجموع ضرب بسط الصنف الذي قام بإستدعاء الدالة في بسط الصنف الآخر أما العدد الثاني فهو نفس الحالة بالنسبة للعدد الاول إلا أنه هذه المرة في المقام في السطر الثالث قمنا بإعادة الصنف المؤقت.

كل الذي قمنا بفعله هو أننا ضربنا بسط الصنف الأول في بسط الصنف الثاني وكذلك بالنسبة لمقامي العددين.

```
Fraction operator/ (Fraction  rhs);
```

كما تعلم فإنه في عالم الرياضيات عند قسمة الأعداد الكسرية فإننا نقوم بقلب الكسر المقسوم عليه ثم ضرب العددين مع بعضهما البعض وهذا ما نقوم به في حالة الصنف Fraction ، حيث أننا أولاً أعلنا عن متغير أطلقنا عليه اسم m في السطر الثالث وفي السطر الرابع قمنا بإسناد بسط الصنف الممرر (صنف المقسوم عليه) إلى المتغير m، وفي السطر الخامس قمنا بإسناد بسط المقسوم عليه (الصنف الممرر) إلى مقام المقسوم عليه، أما في السطر السادس فكما تعلم أن المتغير m يحوي بسط المقسوم عليه وبالتالي قمنا بإسناده إلى بسط مقام المقسوم عليه وهكذا قمنا بقلب العدد الكسري أما بقية الأسطر فهي نفس ما حدث عند زيادة تحميل المعامل *.

```
1     Fraction operator/ ( Fraction rhs)
```

```
2     {
3       int m;
4       m=rhs.GetUp();
5       rhs.up=rhs.GetDown();
6       rhs.down=m;
7       Fraction temp(up* rhs.GetUp() , down* rhs.GetDown());
8       return temp;
9      }
```

Fraction operator+ (int   rhs);

هذه الدالة تقوم بزيادة تحميل العملية + ، لكي يصبح بإمكانك جمع عدد كسري مع عدد طبيعي أو صحيح وليس مع عدد كسري آخر ، ليس هناك الكثير لكي أشرحه ففي السطر الرابع قمنا بإعادة كائن غير مسمى هذا الكائن تم بناؤها بواسطة عددين هما البسط والمقام ولكن هذه المرة قمنا بجمع بسط الصنف مع العدد الممرر إليه. ثم قمنا ببناء الصنف من خلال هذين العددين.

```
1      Fraction operator+ (int rhs)
2  {
4   return Fraction (up+rhs,down);
3  }
```

وبالطبع فهناك بعض المشاكل حول هذه الدالة (وحول الصنف بشكل عام) فلن تستطيع أن تترجم السطر التالي:

```
S=a+S;
```

حيث S صنف من النوع Fraction و a عدد من النوع int ، وأعتقد أنني تناولت هذه المشكلة بشكل عام في الوحدة (اصنع أنماط بياناتك بنفسك) ولا تخف فأنت تجد حلها في الدالة الصديقة في السطر 31.

دوال الإدخال والإخراج:

تستطيع رؤية تصريح هاتين الدالتين في السطرين 29 و 28 وهما ليستا من دالات الصنف ولكنهما صديقتان له وبالتالي فإن جميع الأعضاء الخاصة تعتبر مرئية بالنسبة لهما ، في الحقيقة ليس هناك ما يسمى دوال الإدخال والإخراج ولكني قمت بتسميتها هكذا لتقريب مفهومها لك ، هاتان الدالتان ما هما إلا زيادة تحميل للمعاملين >> و << ، وحتى تفهم هاتين الدالتين فدعني أولاً أعرفك على ما هي الكائنات cout و cin.

في الحقيقة فإن كلمتي cout و cin عبارة عن كائنات تنتمي للتيار أو المكتبة iostream وهي الخاصة بالإدخال والإخراج فالكائن cout ينتمي لـ ostream أي تيار الإخراج أما الكائن cin فينتمي لـ istream أي تيار الإدخال وبالنسبة للمعاملات << و >> فلقد تمت زيادة تحميلهما مثلما تقوم أنت بزيادة تحميل أي معامل ، هذا كل ما أريدك أن تعرفه حتى تفهم الدالتين في الصنف Fraction .

friend ostream &operator << (ostream& ,const Fraction &);

كما ترى الدالة الأولى التي تقوم بزيادة تحميل المعامل << ، تأخذ كبارمترات لها عنوان صنف من النوع ostream والذي هو في هذه الحالة cout والبارامتر الثاني هو عنوان من الصنف Fraction والذي سنقوم

بطباعـته على الشاشــة .لنفرض أنك قمت بإنشاء كـائن مـن النــوع Fraction وقمت بتسميته Example وأردت طبـاعـة ما يحتـويـه هذا العـدد الكسري فإن البديهـي إنك ستكتب هذا السطر التالي:

```
cout << Example ;
```

والذي سيقوم المترجم بترجمـته هـكذا:

```
operator << (cout , c) ;
```

كما ترى ففي الحقيقـة أنك قمت بإسـتدعاء الدالـة ( ) >> operator وقمـت بتمرير الكائن cout إليها والكائن c الذي هــو مـن الصـنف Fraction إلـى هـذه الدالـة ، بعــد ذلـك سـيدخل المتـرجم إلـى جسـم الدالة ( ) >> operator ، والذي هـو لا يتألف حقيقـة إلا من سطر واحد كالتالي:

```
1     return D << temp.up << "/" << temp.down ;
```

كما ترى فإن الدالـة تعــيد الكـائن D والـذي هــو نفسـه الصـنف cout إلا أنها تقـوم ببنـاءه بطريقـة غريبــة حيـث تقـوم بطباعــة العــدد الكســري ، وهـذه الطريقـة هي نفسها الطريقـة التي من الممكن إستخـدامها فـي أي أصـناف أخـرى.

```
friend istream &operator >> (istream& , Fraction &);
```

تقـوم هذه الدالة الصديقـة بإعادة كائن من النـوع istream وهــي تقوم بزيادة تحـميل المعامل << ، ونقوم بتمرير الوسيط cin والوسيط الذي نريد طباعــة العـدد الكسري من خلالهـا ، في السطر الثالث قمنا بالطلب مـن المستخــدم إدخال العـنصر up وبالنسبـة لكلمـة E فهـي نفسهـا cin ، في السطر الرابـع قمنا بالتصريح عـن متغير حرفي ، لا فائدة منه سوى إدخال المعـامل / ، الـذي يميـز بـين الأعــداد الكسريـة وغيرهـا ، وفـي السـطر الخــامس نطلـب مـن المستخـدم إدخال العلامـة أو المتغير c ، في السطر السادس يتأكد البرنامج أن العلامـة أ والحرف المدخلة هـي / وفي حال لم تكـن كذلك ينهار البرنـامج بشكل كـامل ، أو يقـوم بإلقـاء إسـتثناء تسـتطيع أنـت السـيطرة عليـه وإعـادة البرنامج إلى حالتـه الطبيعية ، وفـي الحقيقــة فـإن البرنــامج لا ينهــار وإنمـا يقوم بإلقاء أحد الإستثناءات ويقذفـه إلى نظام التشـغيل ليقـوم بحلـه ، وفـي حال عـدم قدرة الويندوز أو اللينوكس (نظام التشــغيل الـذي أنـت تستعــمله) فإن البرنامج يتوقف عـن العـمل وكل ذلك يتم عبر الكلمــة الأساسيـــة throw ، سوف تتعلم فـي المواضيع اللاحقــة كيـف تتعــامل مـع هـذه المشـاكل ، لا تحـاول إلغـاء السطر السادس ، لأنك إذا قمت بإلغاءه فستقل وثوقيـة الصنف الذي تقوم بكتـابته ، وستجعل من نفسك مهـزلـة حتى وإن كـان صنفك لـيس له مثيل ، والمقارنـة سـتكون شبيهــه بـين الدالة ( )printf والتـي هــي مـن بقايا الســي والتـي لا تسـتطيع حمـاية الأنــواع والكـائن cout . فـي السـطر السابع نطلب من المستخـدم إدخـال مقـام الصـنف وفـي الأخير تعيـد الدالـة الكائن E ، لقـد انتهينا الآن مـن شـرح دوال المعـاملات >> و << وهـذا هــو الهـدف الأسـاسي من هذا التمرين أو المثال وإن كـان الهــدف الأسمــى هــو محاولة توسيع مداركك وإفهـامك كيف تصنع أصنافاً حقيقـيـة يعـتد بها.

```
1. istream& operator >> (istream& E, Fraction& temp)
2. {
3. E >> temp.up;
4. char c;
5. E.get(c);
```

```
6. if (c !='/') throw;
7. E >> temp.down;
8. return E;
9. }
```

قم بتطوير الصنف Fraction:

next phase with Class Fraction:

انتهيت من كتابة هذا الصنف Fraction في غضون أكثر من عشر دقائق بقليل ، وبإمكاني إنهاء 90 % من تطوير هذا الصنف في غضون نصف ساعة أنا لا أفاخر بنفسي ولكن أحاول أن أصور لك مقدار الجهد الذي ستبذله إن قمت بمحاولة تطوير هذا الصنف ، قد تضع أفكاراً جديدة أفضل مني ، ولربما تقوم بصنع نمط بيانات للأعداد الكسرية ينافس النمط float والأنماط الأخرى ، اعتبر تطوير هذا الصنف تحدياً برمجياً وسييسر لك الكثير إن قمت بتطويره بالفعل ، وقد يفتح لك الباب لصنع أنماط جديدة أو حتى أخذ أفكار خلاقة لتصنع بها تطبيقاتك البرمجية ، هذه بعض النقاط التي أعتقد أن الصنف Fraction قد توافقني أو تخالفني فيها الرأي:

- **قم بكتابة دالة بناء أو بالمعنى الأصح معامل تحويل من النمط float إلى الصنف Fraction ، وفكرة هذه الدالة بسيطة حيث تقوم بإسناد النمط flaot أو المتغير إلى المتغير num ثم تقوم بتحويل العدد العشري إلى عدد كسري وإسناد البسط والمقام.**
- **قم بتعريف المعاملات (+) و (-) ونظراً لصعوبتها النسبية أو بالمعنى الأصح غموضها النسبي ، فقم أولاً بصنع دالة جديدة (بشرط أن تكون دالة خاصة) تقوم بتوحيد المقامات أو لربما تجعلها عدداً كسرياً ثم تقوم بتضمينها أو إستدعاءها ضمن دالة المعاملين + و -.**
- **أحد العيوب الأساسية في هذا الصنف Fraction والتي لم أجد لها حلاً لتاريخ كتابة هذا التمرين هو عدم قدرتك على إدخال الصنف Fraction كعدد طبيعي دون كتابة أي مقام (أي تترك للبرنامج إسناد المقام إلى القيمة 1) ، ربما تستطيع حل هذه المشكلة ، والتي حتى وإن وجدت حلاً لها فلن أقوم بتضمينه بل سأدع لك الفرصة أنت لكتابتها والتفكير بها.**



string

السلاسل في لغة السي بلس بلس:

يعتبر التعامل مع السلاسل حسب اللغة c متعباً ومملاً وخطيراً في بعض الحالات وخاصة في حال تجاوز حدود المصفوفة ، لذلك أتت إلينا السي بلس بلس بحل جذري لهذه المشكلة وهي الكائن string ، الذي بإمكانك معاملته وكأنه متغير char إلا أنه يفرق عنه في أنه لا يجب الإعلان عنه كمصفوفة. حتى نستطيع التعامل مع الكائنات string فيجب علينا أولاً تضمين المكتبة string .

بإمكانك الإعلان عن كائن من النوع string كما يظهر من هذا السطر:

```
string STRIG;
```

وليس ذلك فحسب بإمكانك أيضاً إسناد سلسلة إلى سلسلة أخرى كما يظهر من هذا السطر:

```
string S1="Hellow";
string S2=S1;
```

وبالتالي فهذا يمكننا من نسخ سلسلة إلى أخرى دون استخدام التابع strcpy والذي لا يستطيع التعامل مع حالات تجاوز حدود المصفوفة.

أيضاً بإمكاننا دمج سلسلتين في سلسلة واحدة عن طريق المعامل (+)، كما يرى هنا:

```
S2=S1+S2;
```

وليس ذلك فحسب بل بإمكاننا أيضاً أن نبادل سلسلتين ببعضها ، أي نقوم بوضع محتويات السلسلة الأولى في السلسلة الثانية ونضع محتويات السلسلة الثانية السابقة في السلسلة الأولى ، بواسطة التابع swap الذي يتبع كائنات string ، انظر لهذا المثال:

```
S1.swap(S2);
```

الآن سنقوم بكتابة مثال كودي يحوي أساسيات مميزات هذا الكائن string ، انظر إلى هذا الكود:

CODE

```
1. #include <iostream>
2. #include <string>
3. using namespace std;

4. int main()
```

```
5. {
6. string S1= "languge Java";
7. string S2= "Languge C++";

8. cout <<"string1:\t\t" << S1 << endl;
9. cout <<"string2:\t\t" <<  S2 << endl;

10.      cout << "After swaping" << endl;
11.      S1.swap(S2);

12.      cout <<"string1:\t\t" << S1 << endl;
13.      cout <<"string2:\t\t" <<  S2 << endl;

14.      S2=S1+S2;

15.      cout <<"S2=S1+S2:\t\t" << S2 << endl;
16.      return 0;
17.      }
```

وسيكون ناتج هذا الكود كما يلي:

```
string1:       languge Java
string2:       Languge C++
After swaping
string1:       Languge C++
string2:       languge Java
S2=S1+S2:            Languge C++languge Java
```

الآن عليك محاولة فهم الكود السابق لأني شرحت أغلب ميزات الكائن string في الأسطر السابقة.

الإدخال والإخراج مع كائنات string :

تستطيع التعامل مع الإدخال بواسطة الكائن cin ، إلا أن المشاكل السابقة ستكون موجودة وعليك التعامل معها ، أما الإخراج فيكون بواسطة الكائن cout .

يوجد تابع مستقل اسمه getline ، يأخذ هذا التابع وسيطين الأول هو الكائن cin والوسيط الثاني هو الكائن string والوسيط الثالث هو حرف الانهاء ولا تحتاج أنت لكتابة الوسيط الثالث فهو سيكون افتراضياً الحرف '\n'.

الآن انظر لكيفية إدخال الكلمات إلى السلسلة S1 :

CODE

```
1. #include <iostream>
```

```
2. #include <string>
3. using namespace std;
4.
5. int main()
6. {
7.
8.    string S1= "languge Java";
9.
10.          getline (cin , S1,'\n');
11.          cout << S1;
12.          return 0;
13.   }
```

كما ترى في السطر 10 فإن التابع getline ، يأخذ كوسيط أول له الكائن cin ، قد تتساءل عـن غرابة هذا الإجراء ولكن لا عليك فحينما تتقـدم خـلال مواضيـع البرمجة الشيئية ستعرف ماذا يعـني كل هذا الكلام، المهـم الآن أن تعلـم أن التــابع getline ، إذا مــا أردت إدخــال سلســلة فعليـك بوضـع cin كوســيط وستفهـم حينما تتقدم في البرمجة كيف يعـمل هذا التابع.

إيجاد كلمة ما ضمن سلسلة:

ربما أنك تبحث عـن كلمة ضمن سلسلة وتريد أن تعلم موقعها بالضبط ، فكـل ما عليك هـو استخدام التابع العضو find ، وستجد أين هـو موضع تلك الكلمـة انظر إلى هذا المثال الكودي البسيط:

CODE

```
1. #include <iostream>
2. #include <string>
3. using namespace std;
4.
5. int main()
6. {
7.
8.    string S1= "languge Java";
9.
10.          int x=S1.find("Java");
11.          cout << x<< endl;
12.
13.          return 0;
14.   }
```

يقوم التابع find بعد الأحرف (بما فيها المسافات) حتى يجد الكلمة Java وذلك في السطر 10 وحينما يجد الكلمة Java فإنه يقوم بوضع عدد الأحرف التي عدها في المتغير x ثم في السطر 11 يطبع الموضع الذي وجده ، والذي سيكون 8 ، لتتأكد من ذلك قم بالعد من بداية السلسلة ابتداءً من الصفر وليس الواحد حتى أول حرف في الكلمة Java وهو الـ J وستجد أنه بالفعل 8 .
أيضاً بإمكانك معرفة حجم السلسلة وكم حرف موجودة فيها عن طريق التابع size( ) ، فبإمكانك معرفة حجم السلسلة S2 كما هو ظاهر في هذا السطر:

```
int n=S1.size();
```

حيث الآن سيصبح المتغير الرقمي n يحوي حجم السلسلة أو عدد حروفها (لا فرق هنا بين الحجم وعدد الحروف فكما تعلم أن char عبارة عن بايت واحد وليس بايتين أو ثلاث حتى نقول أن هناك فرق).
ليس ذلك فحسب بل بإمكانك أيضاً الوصول إلى أي حرف في السلسلة ، كما تصل إلى أي عنصر من عناصر المصفوفة فللوصول إلى الحرف الثاني في السلسلة S2 تستطيع كتابة هذا السطر:

```
char x= S2[1];
```

والسبب في وضعنا الرقم 1 هو أن رقم العناصر في أي مصفوفة يبدأ من الصفر وليس من الواحد.

نسخ السلاسل:
هناك طريقة أخرى أيضاً لنسخ سلسلة إلى سلسلة أخرى ، وهي عن طريق التهيئة ، بإمكانك تهيئة سلسلة بسلسلة أخرى ، انظر إلى هذا السطر:

```
string s1(s2);
```

ليس ذلك فحسب بل بإمكانك تهيئة سلسلة بجزء من سلسلة أخرى.
لنفرض أنك تريد تهيئة سلسلة بأول ستة أحرف من سلسلة أخرى ، انظر إلى هذا السطر:

```
string S3(S1,0,6);
```

في هذا السطر يتم نسخ أول ستة أحرف من السلسلة S1 ، إلى السلسلة S3 ، الآن انظر إلى دالة البناء للكائن S3 ، الوسيط الأول عبارة عن السلسلة التي نود تهيئة الكائن بها ، الوسيط الثاني هو العنصر الذي نود بدأ النسخ منه وهو في حالتنا هذه العنصر الأول (0) أي بداية السلسلة ، إذا كتبت 1 فسيبدأ البرنامج النسخ من الحرف الثاني وهكذا ، أما الوسيط الثالث فهو عدد الأحرف أو العناصر التي نود نسخها.

التابع substr( ) :
هناك أيضاً تابع يقوم بنفس مهمة تابع البناء السابق وهو التابع substr( ) .
يستقبل هذا التابع وسيطان ، الوسيط الأول هو رقم الحرف الذي تود بدأ النسخ منه والوسيط الثاني هو عدد الأحرف أو العناصر التي تود نسخها ابتداءً من الوسيط الأول. انظر إلى هذا السطر:

```
S2= S1.substr ( 5,9);
```

سيأخذ البرنامج 9 أحرف من السلسلة S1 ليس من أول السلسلة بل ابتداءً من العنصر الخامس فيها ويقوم بنسخها إلى السلسلة S2 .

التابعان ( )begin و ( )end :

هـناك أيضاً تابعان بسيطان يعيد التابع begin العـنصر الأول أما التـابع ( )end فيعـيد الحرف الأخير ، بإمكانك تهيئة السلسلة هـكذا:

```
string S2(S1.begin() , S1.end() );
```

التابع ( )Capacity :

حينما تقوم بإنشاء سلسلة فإن المترجم يحجـز لها ذاكـرة ليسـت فـي نفـس عـدد الأحـرف التـي أدخلتها بـل أكبـر قليلاً والسـبب فـي ذلـك حتـى يصـبح بإمكانـك إضـافة أحـرف قليلـة دون أن يقـوم المتـرجم بإلغـاء ذاكـرة الأحـرف السابقة وتخصيص ذاكرة جديدة تضم الأحرف التي أدخلتها والأحرف السابقة ، فهذه هي طريقة عـمل الكائن string ، تقوم السلـسلـة فـي أغلـب الأحيـان بحجز 31 حرف حتى لو أدخلت حرفاً واحـد فحسـب ، ثـم إذا أضـفت 20 حـرف فسيتم إدخالها دون مشاكل ودون تخصـيص وإعـادة تخصـيص للـذاكرة ، لكـن ماذا لو قررت زيادة الأحرف عـن 31 حينها سـيتم تخصـيص وإعـادة تخصـيص للذاكرة حتـى تسـتطيع السلسـلة التعامل مـع هـذه المشـكلة ، تعـرف هـذه الأحـرف الزائـدة بأنها قـدرة المصـفوفة وحتـى تعلـم قـدرة السلسـلة علـى التخزين دون حدوث تخصيص وإعادة تخصيص فبإمكانك طباعة القيمة العائدة للتابع العضو ( )capacity .

مزيد من التوابع ( )append و ( )insert .... :

التابع append يضيف سلسلة إلى نهاية السلسلة أو يقوم بتـذييل السلسـلة بسلسلة أخرى أما التابع insert فهـو يضيف سلسلة إلى إلى أي موقع تريده من السلسلة.

الآن سنستعرض مثالاً عـملياً يقوم بتناول أغلب هذه التوابع.

CODE

```
1. #include <iostream>
2. #include <string>
3. using namespace std;
4.
5. int main()
6. {
7.    string S1= "a lot of programmers" ;
8.    cout << "Sting S1\t\t" << S1 << endl;
9.    cout << "S1.size\t\t\t"  << S1.size() << endl;
10.         cout << "S1.capacity()\t\t" << S1.capacity() << endl;
11.
12.         cout << endl;
13.
14.         S1.append(" love this languge");
15.         cout << "Sting S1\t\t" << S1 << endl;
16.         cout << "S1.size\t\t\t"  << S1.size() << endl;
17.         cout << "S1.capacity()\t\t" << S1.capacity() << endl;
```

```
18.
19.         cout << endl;
20.
21.         S1.insert(0,"C++ Languge ");
22.         cout << "Sting S1\t\t" << S1 << endl;
23.         cout << "S1.size\t\t\t"  << S1.size() << endl;
24.         cout << "S1.capacity()\t\t" << S1.capacity() << endl;
25.
26.         return 0;
27.    }
28.
29.
```

مخرجات هذا الكـود ، هي كالتالي:

```
1. string S1            a lot of programmers
2. S1.size              20
3. S1.capacity()        31
4.
5. string S1            a lot of programmers love this lanbuge
6. S1.size              38
7. S1.capacity()        63
8.
9. string S1            C++ a lot of programmers love this lanbuge
10.    S1.size              50
11.    S1.capacity()        63
```

- لقد قمنا في هذا الكـود بالإعلان عـن السلسلة S1 في السـطر 7 ، سـنقوم خـلال ثـلاث مراحـل بإضـافة سلاسـل إضـافية إلـى هـذه السلسلة بطرق مختلفة.
- فـي السـطر 8 قمنـا بطباعــة محتويـات هـذه السلسـلة أمـا فـي السطر 9 فلقد طلبنا طباعـة حجم البرنامج أما في السطر 10 فلقد طلبنـا مـن البرنـامج طباعـة قـدرة السلسـلة علـى التخـزين قبـل تخصيص وإعادة تخصيص الذاكرة.
- في السطر 14 قمنا باستخدام التابع append ، والذي قمنـا بتمرير سلسـلة كوسـيط لـه حيـث سـيأخذ هـذه السلسـلة ويـذيل بهـا السلسلة S1 أو بمعنـى أوضح يقوم بوضعها فـي نهايـة السلسـلة S1 ، في السطر 15 قمنا بطباعة محتويات السلسـلة بعـدما قمنا بتذيليها ، وفـي السـطر 16 قـام البرنـامج بطباعـة حجـم السلسـلة والذي حالياً تجاوز قدرة السلسلة على التخزين حيث تجـاوز العـدد 31 ليصـبح 50 حرفـاً ، سـيقوم الكـائن string بتخصـيص وإعـادة تخصيص الذاكرة حتى أصبح حجم السلسلة 63 .

- في السطر 21 استخدمنا التابع insert والذي يأخذ وسيطين له ، الوسيط الأول هو الموقع الذي تود الإضافة ابتداءً منه أما الوسيط الثاني فهو السلسلة التي تود إضافتها ، في الأسطر 21 و 22 و23 قمنا بطباعة محتويات السلسلة وخصائصها كالحجم والقدرة ، لاحظ أن القدرة لم تختلف عن آخر إضافة بالتابع append .

تابع الاستبدال بين سلسلتين replace( ) :

قد تود في بعض الحالات البحث عن كلمة معينة في سلسلة ما واستبدالها بكلمة أخرى ، يوفر لك الكائن string ، تابعاً يقدم لك هذه الخدمات هو التابع replace( ) ، حيث يأخذ ثلاث وسائط ، الوسيط الأول هو مكان العنصر الذي تود وضع السلسلة فيه ، الوسيط الثاني هو حجم الكلمة التي تود إلغائها ، الوسيط الثالث هو السلسلة التي تريد وضعها بدلاً من ذلك الحجم.
لاحظ هنا أنه يجب عليك تحديد حجم السلسلة أو الكلمة التي تود استبدالها إذا كانت الكلمة التي تود استبدالها مؤلفة من حرفين وكانت الكلمة التي تود وضعها بدلاً عنها مؤلفة من 20 حرفاً فسيتم إلغاء الكلمة المؤلفة من حرفين ووضع بدلاً عنها الكلمة المؤلفة من 20 حرفاً وبالطبع سيزيد حجم السلسلة ، انظر إلى هذا المثال:

CODE

```
1. #include <iostream>
2.
3. #include <string>
4. using namespace std;
5.
6. int main()
7. {
8.    string S1("The Java Programming Languge");
9.    cout << "S1 Befor\t\t" << S1 << endl;
10.
11.         int p=S1.find("Java");
12.
13.         string S2(S1,p,4);
14.
15.         cout << "S2\t\t\t" << S2 << endl;
16.
17.         S1.replace(p,S2.size(),"C++");
18.
19.         cout << "S1 NOW \t\t\t" << S1 << endl;
20.
21.         return 0;
22.    }
```

الجملة التي لدينا هي The Jave Programming Languge نود استبدال كلمة Java ووضع بدلاً عنها كلمة C++ .

- في السطر 8 قمنا بالإعلان عن سلسلة S1 تحوي الجملة السابقة ، قمنا بطباعتها في السطر 9.
- في السطر 11 يقوم البرنامج بالبحث عن الكلمة Java وتخزين موقعها لدى المتغير p .
- في السطر 13 قمنا بالإعلان عن سلسلة S2 والتي تقوم بنسخ كلمة Java الموجودة في السلسلة S1 وإسنادها إليها، والسبب في قيامنا بهذا الإجراء هو حتى نعرف كلمة Java حتى نستخدمها كوسيط للتابع replace ، قد تستغني وتقول أن حجمها هو 4 وبالتالي لا داعي لمثل هذا الإجراء ولكن من الأفضل اعتماد هذه الطريقة لأنك في المشاريع الكبيرة لن تشغل نفسك بعد الأحرف وخاصة إذا كانت ليست كلمة بل جملة ، أضف إلى ذلك أنك قد تخطيء في العد.
- يقوم السطر 15 بطباعة السلسلة S2 ، حتى تتأكد بالفعل أنها تحوي الكلمة Java .
- التابع replace يظهر في السطر 17 ، حيث يأخذ ثلاث وسائط الوسيط الأول هو موقع الاستبدال وهو في هذه الحالة المتغير p ، الوسيط الثاني هو حجم السلسلة أو الكلمة التي نود استبدالها وهي في هذه الحالة S2.size( ) ، أما الوسيط الثالث فهي الكلمة التي نود وضعها بدلاً من الكلمة Java وهي التي في هذه الحالة الكلمة C++ .
- يقوم السطر 19 بعرض السلسلة S1 وسترى أن البرنامج نجح في الاستبدال وأصبحت السلسلة هكذا: The C++ Programming Languge .

تابع المسح erase( ) :

هناك تابع آخر مهم هو التابع erase يستقبل هذا التابع بارامترين اثنين ، الأول هو المكان الذي تود بدأ المسح منه ، والبارامتر الثاني هو عدد الأحرف التي تود مسحها ابتداءً من البارامتر الأول.
انظر إلى هذا المثال الكودي حتى تفهم المقصود، سنقوم في هذا المثال بتعديل محتويات الكود السابق وسنستخدم التابع erase دون التابع replase .

CODE

```
1. #include <iostream>
2. #include <string>
3. using namespace std;
4.
5. int main()
6. {
7.     string S1("The Java Programming Languge");
8.     string S2="C++";
9.     cout << "S1 Befor\t\t" << S1 << endl;
```

```
10.         cout << "S2\t\t\t" << S2 << endl;
11.
12.         int p=S1.find("Java");
13.
14.         S1.erase(p,4);
15.         S1.insert(p,S2);
16.
17.         cout << "S1 NOW \t\t\t" << S1 << endl;
18.
19.         return 0;
20.   }
```

- في السطر 14 سيقوم البرنامج بمسح الكلمة Java من البرنامج بواسطة التابع العضو erase حيث يأخذ في البارامتر الأول موقع بداية المسح وفي البارامتر الثاني عدد الأحرف التي سيمسحها.
- في السطر 15 يتم وضع السلسلة S2 التي تحوي الكلمة C++ (كما هو ملاحظ في السطر 8) في السلسلة S1 وفي المكان الذي كانت توجد به كلمة Java .
- مخرجات البرنامج هي نفسها التي في الكود السابق ، ولكن كما ترى فإن الكود السابق أكثر سهولة وأكثر فعالية ولكن هذا لا يمنع من أنك تحتاج كثيراً للتابع erase( ) في تطبيقات أخرى.

حجم الكائن string( ) :

كما رأيت فإن الكائن string حجمه أكبر من عدد الأحرف المخزنة فيه كما يحدد لك التابع capacity( ) الحجم الصحيح للكائن ، وحتى إن قمت بتجاوز حجم الكائن فستتم عملية تخصيص وإعادة تخصيص للذاكرة ، حتى عند حد معين حينها يتوقف string عن التخصيص وإعادة التخصيص وينهار برنامجك وحتى تعلم متى يتوقف الكائن عن التخصيص وإعادة التخصيص فاستخدم التابع max_size( ) ، انظر إلى هذا المثال:

CODE

```
1. #include <iostream>
2. #include <string>
3. using namespace std;
4.
5. int main()
6. {
7.    string S1("When it stop");
8.
9.    cout << "S1 Now\t\t" << S1 << endl;
10.         cout << "S1\t\t" << S1.max_size() << endl;
```

```
11.
12.          return 0;
13.    }
```

سيطبع لك هذا البرنامج محتويات السلسلة S1 ، والحجم الذي لن تتم من بعدها إعادة أو زيادة تخصيص للذاكرة ، وهو في هذه الحالة 4294967293 حرف أو عنصر حسب ما يقوله مترجم Visual C++ ، بالطبع يختلف الأمر عن بقية المترجمات.

الوراثة

Inheritance

بدايـة:

الوراثة أحـد أهـم مبادئ البرمجـة الكائنية وهـي تحـظى بدعـم كبيـر مـن السـي بلس بلس . هل تتذكر الهـدف من البرمجـة الشيئيـة؟
هذا الهـدف والذي هـو محاولة تمثيل العالم الواقعـي تقوم الوراثـة بتحقيقـه أو على الأقل السير خـطوات تجاه تحقيق هذا الهـدف.

الفرق بين الوراثة في العالم الحقيقي وعالم البرمجـة:

لنفهـم أولاً معـنى الوراثـة فـي العـالم الحقيقـي وسنفهـمـه علـى مستــوى الحيوانات ، كما تفهـم فإن الأسـد يقوم بتوريث صفاتـه إلى الشبل.. والذي قام هـنا بفعل الوراثـة هـو الأسـد أي الأب أو بمعـنى برمجـي الصـنف الأسـاس.. أما الشبـل أو الصـنف المشـتق فهـو الـذي يأخـذ مـن الأب أي أنـه يسـتقبل. وهذا هـو المعـنى البديهي للوراثـة.
إلا أن الوضـع يختلـف بالنسـبة للوارثـة فـي عـالم البرمجــة . فالـذي يقـوم بالوراثـة هـنا ليس الأب ولكن الابن... لا تحـاول فهـم الوراثـة علـى أن هـناك صنف أساس يقوم بتوريث صفاته إلى أبناءه بل افهـمـه على مبدأ أن الصـنف الذي تنشأه صنف مستقل عـن جميع الأصناف الأخـرى، وأنـه هـو الذي يحـدد خياراتـه وشكلــه فبإمكـانــه أن يتـوراث مـن أي صـنف يريـد بشـرط أن يخــدم الغرض من إنشاء هذا الصـنف ، وبإمكـانــه أيضـاً أن يتـوارث مـن عــدة أصـناف دفعـة واحـدة ولـيس صـنف أو صـنفين فقـط وهـذا مـا نطلـق عليـه بـالتوارث المتعـدد ، وعـموماً ينقسـم التوارث إلى قسـمين

1- **التوارث العام : وفيـه يملـك الصـنف الابـن الصـنف الأب بجميـع أعضـاءه الخاصة والعامـة.**
2- **التوارث الخاص: وفيه يملك الصـنف الابـن طريقـة الإسـتخـدام للصـنف الأب ، فجميع أعضاء الأب تتحـول إلى أعضاء خاصـة لدى الصـنف الابـن في حال التوارث الخاص.**

مبدأ التجريد:

لتستفيد أعـلى استفادة من ميزات الوراثـة فعليك إستعـمال هذا المبدأ دائماً في جميع أصنافك ، فهذا المبدأ يحفظ لك الوقت والجهـد، ويزيد من الإنتاجيــة ومن ميزة إعادة الإستخـدام.
لنفرض أنـه طلـب منـك إنشـاء برنـامج تسـجيل الطـلاب فـي الجامعــة هـذا الشـهر وفي الشـهر القادم ستقوم بإنشاء برنامج إدارة الموظفين لشركـة مـا. بالطبع فإن أول ما تفكر فيه هـو أنك ستقوم بإنشاء البرنامجين كل على حـدة ، ولكن الوراثـة مع مبدأ التجريد هــي التـي تعـطيك إمكـانية الإسـتفادة مـن مزايا البرمجـة الكائنيـة ، فبدلاً من أن تقوم بإنشـاء صـنف طالـب فـي برنـامج تسـجيل الطلاب وإنشاء صنف موظف في برنامج إدارة الموظفين.. لما لا تقوم بتجريد هـذان الكائنـان وتنظر لهـما ليس على أساس أنهـما موظفين أو طـلاب بل علـى أسـاس أنهـم أشخـاص.. وبالتالي تقوم بإنشاء صنف اسمـه شخـص.. ثـم تـأتي بعـد ذلـك بالصـنفين الطالـب والموظـف وتقـوم بتوريثهــما صـنف الشخـص. وليس ذلك فحسب بل تقوم بإنشاء صـنف اسمــه طالـب جامعــي تقـوم بتوريثــه صـفات الأب الصـنف الطالـب. أيضـاً حينمـا تقـوم بإنشـاء صـنف

تسجيل الطلاب وصنف آخر تسجيل الموظفين. فلماذا لا تقوم بتجريدهما والنظر للصنف على أنه صنف تسجيل الأشخاص مثلاً.. قد تستغرب ما أقول ولكن لنفرض أنه بعد مدة معينة طلب منك إنشاء برنامج تسجيل الطلاب العسكريين فحينها لن تقوم بإعادة ما كنت تقوم بفعله بل كل ما عليك هو الرجوع إلى مكتبة الأصناف التي تملكها والتي قمت بإنشاءها مسبقاً وتقوم إما بإضافة أصناف جديدة أو جعل تلك الأصناف في خدمتك ، وبالتالي تزيد من إنتاجيتك بدلاً من أن تقوم بإنشاء البرنامج من الصفر.
ولنفرض أنه طلب منك إنشاء برنامج ATM فحينها سترجع إلى الصنف الشخص وتقوم بإنشاء هذه الأصناف: عميل ، مدير فرع بنك ، موظف بنك وتقوم بتوريثهم صنف الشخص.
مبدأ التجريد أوسع مما ذكرت ويستخدم في مواضيع أخرى غير الوراثة.
كل الذي أريدك أن تتعلمه أن تنظر إلى الأصناف التي تقوم بإنشاءها بتجريد أكثر وليس نظرة المستعجل على إنشاء برنامج.

الفرق بين الوراثة والنسخ واللصق:

من البديهي أن تعتقد أن الفائدة الوحيدة للوارثة ، هي إعفاء المبرمج من إعادة كتابة صنف كامل وأنه بإمكانك التخلص من هذه الإشكالية بواسطة أدوات محرر النصوص عبر نسخ النصوص ثم لصقها ؛ بالرغم من صحة هذا الكلام جزئياً إلا أن الوراثة تعطيك فائدة أكبر وأكثر من مجر إعفاءك من إعادة الكتابة ، فإذا افترضنا أن الصنف الأب (أ) والذي له ثلاثة أبناء وهم (ب) (جـ) (د) قد وقع فيه أحد الأخطاء فإنك لن تضطر إلى تعديل الخطأ في جميع الأصناف بل في صنف واحد فقط هو الأب ، أيضاً الوراثة تمنحك رؤية أكثر دقة عند تصميم برنامج معين فمخطط الـ UML يكون أفضل وأكثر بساطة من رؤية أصناف ليس بينها أي وراثة.
أيضاً الوراثة تمنح الأصناف التي تصنعها وثوقية أكثر خاصة إذا قمت بإشتقاقها من أصناف تم التأكد من عدم حصول أي خطأ فيها.
بالطبع أنت لن تستفيد بهذه المزايا إلا حينما تقوم بعمل برامج قوية وليس برامج بسيطة.

إشتقاق الأصناف:

سنقوم الآن بإنشاء مثال كودي ، هذا المثال ليس له فائدة وإنما يعرفك على الوراثة فحسب على الصعيد الكودي:

CODE

```
4. #include <iostream.h>
5.    class Father
6.    {
7.    protected:
8.    int itsAge;
9.    public:
10.         Father():itsAge(8)
11.         { cout <<"\n the Father ALIVE \n" ; }
12.         ~Father() {cout << "\nthe Father DIEEEE" ; }
13.   GetitsAge(){ return itsAge ; }
```

```
14.   };
15.
16.   class son: public Father
17.   { public:
18.   son() { cout << "\nthe son is ALIVE\n";}
19.   ~son() { cout <<"\nthe son die \n" ; }
20.   };
21.
22.   void main()
23.   {
24.         son you;
25.         cout << endl << you.GetitsAge();
26.   }
```

وكما ترى فإن ناتج البرنامج هـو كالتالي:

ناتج الكــود

```
1.    the Father ALIVE
2.    the son is ALIVE
3.    8
4.    the son die
5.    the Father DIEEEEEEE
```

الصنف Father عبارة عـن صنف يملك متغير عـددي وله دالتين إحداهـما دالة البناء ودالة أخرى للوصول إلـى العــنصر المخفــي ، وكمـا تـرى فـإن المتغيـر العـددي في الصنف itsAge لـم يوضـع فـي القسـم الخـاص بـل فـي القسـم المحـمي ، كما هـو موجـود فـي السـطر الرابـع ، والسـبب فـي ذلـك أنــه إذا جعلنا المتغير itsAge في الوضع الخاص فإن الصنف الابن son لن يتمكن مـن رؤيته بالرغــم مـن أنــه قـد حصـل عليهـا بواسطــة الوراثــة وهـذا يعـود فـي الحقيقــة لمســتوى الحمــاية للمتغيـر itsAge فهــو لا يســمح حتـى للأبنـاء برؤويتـه ولتمكن الأصناف الأبناء من رؤوية الأعضاء الخاصة فكل ما عليك هـو جعلهم في المستوى المحـمي .

في هذا المثال لن يضير وضع المتغير itsAge في القسم الخاص لأنك وضعت له دالة وصول.

فـي السـطر 13 قمنـا بإنشـاء الصـنف son والـذي يتـوارث الصـنف Father والطريقة الكـودية لفعل ذلك هـي:

```
27.   class son          :       public          Father
          اسم الصنف        نقطتين    نوع التوارث       الصنف المشتق
```

كما ترى فلقد فصلنا بين نـوع التوارث والتصريح عــن الصـنف بنقطتيـن ونــوع التوارث الذي لدينا هـو عام public ثم كتبنا اسم الصنف المشتق

دوال الهـدم والبناء:

كما تلاحظ في المثال السابق فلقد عرفنا دالتي بناء الصنفين الصنف الأساس والصنف المشتق لتطبع عبارة تخبر عن إنشاءها وكذلك دالتي الهدم جعلناها تخبر عن هدم الصنف الذي يحتويها ، كما ترى في السطر 21 قمنا بالإعلان عن كائن من الصنف المشتق اسمه you ، ثم تنتهي الدالة main( ) انظر لناتج السطر 21 فلقد استدعينا دالة بناء الصنف القاعدة ثم دالة بناء الصنف المشتق وحينما تم تهديم الكائن you تم استدعاء دالة هدم الصنف المشتق ثم دالة هدم الصنف الأساس.

مثال على مبدأ الوراثة:

لن نعمد حالياً إلى تطوير أمثلة ذات قدرة عالية ، بل سنركز على أمثة بسيطة الغرض منها إيصال المعلومة وليس إثراءها وأفضل وسيلة لفعل ذلك هي جعل دوال البناء تقوم بكتابة ما يدل على إنشاءها سنأتي الآن بأحد الأمثلة

CODE

```
1. #include <iostream.h>
2.
3. class father
4. {
5. public:
6. father()
7. { cout << endl <<"I am Alive";}
8. father(int x)
9. {
10.    cout << endl << " I am Alive (int) " ;}
11.
12.    };
13.
14.    class son : public father
15.    {
16.    public:
17.    son(){ cout <<"\n Hellow son\n " ;}
18.    son(int y): father(y)
19.    {
20.    cout << "\n Hellow son (int)\n" ;}
21.    } ;
```

كما ترى في الصنفين السابقين فلقد أنشأنا صنفين اثنين وقمنا بزيادة تحميل دوال البناء لهما ، فهناك دالة البناء الإفتراضية وهناك دالة البناء التي تستقبل عدداً من النوع int سنقوم الآن بكتابة هذا السطر في الدالة main( ) ونرى مالذي سوف يحدث:

```
1    son ;
```

سيكون الناتج بشكل طبيعي كالتالي:

```
1    I am alive
```

```
2    Hellow  son
```

الذي حـدث هـو أن المتـرجم قـام بإسـتدعــاء دالـة البناء الإفتراضـية الخاصــة بالأب ثم دالة البناء الإفتراضية الخاصـة بالابـن.

الآن لنرى مالذي سيحدث إذا قمنا بكتابة السطر التالي:

```
1     son(5);
```

حتى نفهـم مالذي سيحدث لننظـر رؤيـة حـول مالـذي سيسـتدعيه المتـرجم؛ وهو كالتالي:

```
22.   son(int y): father(y)
23.   {
24.   cout << "\n Hellow son (int)\n" ;}
```

في السطر 22 قمنا بتهيئـة الدالة بدالة بناء الأب وقمنا بتمرير نفس القيمــة لها والتي هـي 5 ، وهذا يحدث في قسم التهيئـة ، ثم يتحــول البرنـامج إلـى دالة البناء التي تستقبل عـدد فـي الصـنف الأب ويتجاهـل الدالـة الإفتراضيــة وبعـد ذلك يدخل في تنفيذ دالة البناء الخاصـة بالابن.

خلاصـة إستدعاء دوال البناء عـند التوارث:

-لنفترض أن لدينا صنف أساس هـو A وهـناك الصنف الابن وهـو a.

- لكـل مـن الصـنفين دالتـي بنـاء إحداهــما هــي الدالـة الإفتراضـية والدالـة الأخـرى تستقبل عـدداً من النـوع int.

-إذا قمت بإنشاء كائن من الصنف a فإنـه ابتداءاً يستدعــي الدالة الإفتراضية الخاصـة بالأب ثم يستدعــي دالة البناء الخاصـة بـالابن.

- إذا قمت بإنشاء كائن مـن الصـنف a ومـررت لــه عـدداً مـن النــوع int ، فلـن يقوم بإستدعاء دالة البناء التي تستقبل عدداً من النوع int الخاصة بالأب، بل سيستدعــي دالة البناء الإفتراضية (التي لا تستقبل أعـداد) الخاصـة بالأب ثم يقوم بإستدعاء دالة البناء الخاصـة بالابن والتي تقوم تستقبل أعـداد.

-لجعل المترجم يقوم بإستدعاء دالة بناء الصنف الأي التي تستقبل أعـداد فلا بد عليك من تغيير تعريف دالة البناء الخاصـة بالابن التي تستقبل عـدداً وتضـع في جزء التهيئـة إستدعاء لدالة الأب الخاصـة بإستقبال أعـداد هـكذا:

```
1     a (int x): A(int x) { }
```

-لا تقم بوضع الإستدعاء داخل جسم دالة البناء الخاصـة بـالابن فهـذا سـيقوم بإستدعاء دالـة الأب الإفتراضـية أولاً ثـم يقـوم بدخــول جسـم دالـة بنـاء الابـن ويستدعـي دالة بناء الأب الخاصة بإستقبال الأعـداد.

حتى تفهـم بشكل أفضل طريقة تنفيذ دوال البناء في الكائنات المتوارثـة:

قم بإختبار المثال السابق عـدة مرات ولجميـع الحـالات ، حتـى تفهــم مالـذي حدث بالضبط.

بعـد أن انتهينا من هذه المواضيع (مواضيع دوال البناء والهـدم) ، فإنك بالتأكيد ترغب في أحد الأمثلـة العمليـة والمثـال الـذي سنقدمــه لـك ، سـيكون مثـالاً رسومياً لا أعـني أننا سنقوم برسم أشكال ثلاثية الأبعاد بل أشـكال بسيطــة جداً للغاية ، الغرض منها محاولة تطبيق ما تعلمناه على أرض الواقع.

CODE

```
1. class shape
2. {
3. protected:
```

```
4. int itsX1;
5. int itsX2;
6. public:
7. shape();
8. shape(int ,int);
9. void Draw();
10.   };
11.   shape::shape():itsX1(5),itsX2(6)
12.   {}
13.   shape::shape(int x,int y)
14.   {
15.   itsX1=x;
16.   itsX2=y;
17.   }
18.   void shape::Draw()
19.   {
20.   for (int d1=0;d1<itsX1 ;d1++)
21.   {
22.   for (int d2=0;d2<itsX2;d2++)
23.   cout << "*";
24.   cout <<endl;
25.   }
26.   }
27.
28.   class square:public shape
29.   {
30.   public:
31.         square(int x):
32.           shape(){itsX1=itsX2=x;}
33.   };
```

من المفترض أن يكون الصنفين shape و squere مفهومان لديك على أقل تقدير ؛ كما ترى فإن الصنف shape هو الأساس وله دالتي بناء إحداهما إفتراضية والأخرى تستقبل أبعاد الشكل المراد رسمه أما الصنف الابن square فهو يستقبل عدد واحد فقط وهو طول الضلع ليقوم برسم المربع وبقية الدوال والمتغيرات يتوارثها عن الصنف الأساس ، بقي لدينا الآن هو كيفية تنفيذ هذه الأصناف ، وبالطبع سنكتب الدالة main( ) ولكن لن نستخدم فيها إلا الصنف square

CODE

```
void main()
```

```
1. {int x;
2. do
3. {
4. cin >> x ;
5. cout << "\n";
6. squere A(x);
7. A.Draw();
8. } while (x!=0) ;
9. }
```

وكما ترى أيضاً فإن رسم المربعات التي تريدها سيستمر حتى تقوم بإدخال الرقم صفر لطول ضلع المربع ثم يتوقف البرنامج عن العمل تستطيع بنفس الوقت إستخدام كائنات الصنف shape ولكن هذه المرة ستستخدم قيمتين لتمريرها إلى الكائن والبقية لديك معرفة ولا تحتاج لشرح.

تجاوز دالات الصنف الأب:

سنقوم الآن بإنشاء صنف جديد هو Triangle (مثلث) وسنقوم بالتوارث من الصنف الابن square ولكن سنرى إحدى المشاكل وهي كيفية التعامل مع طريقة رسم المثلث ، فالطريقتين لدى الصنفين السابقين هي واحدة ولكن بالنسبة للمثلث فهي تختلف ، الحل الوحيد هو إنشاء دالة تقوم بفعل ذلك وهي إما بكتابة دالة جديدة أو بتجاوز الدالة الأساسية للكائن ، قبل إنشاء هذا الصنف الجديد فلا بد علينا معرفة الدوال التي نرغب في تجاوزها والدالة التي سنقوم بتجاوزها هي الدالة Draw( ) ، سنضع هذه الدالة في جسم تعريف الصنف لأننا نرغب في تجاوزها وأيضاً لا بد علينا من إعادة تعريف دالة البناء لأنه يجب أن يكون لديك دالة بناء ، وذلك بسبب أنها معرفة في الدالة الأساس وبالتالي فإن المترجم سيعتبرها معرفة لدى الدالة الابن ونظراً لأنه لن يجدها فسيعطيك أحد الأخطاء:

CODE

```
1.  class triangle :public squere
2.    {
3.    public:
4.     triangle(int x)
5.          :squere( x){}
6.    void Draw();
7.
8. };
9.
10.   void triangle::Draw()
11.   {
12.         for (int d1=0;d1<itsX1+1;d1++)
13.         {     for (int d2=0; d2<d1;d2++)
14.                     cout <<"*";
```

```
15.         cout << endl;}
16.    }
```

لقد قمنا بإعادة تعريف دالة البناء والسبب مذكور في الصفحة السابقة ، وبالطبع لن نحتاج لشرح دالة البناء أما بالنسبة للدالة Draw( ) ، فلقد قمنا بالتصريح عنها في جسم تصريح الصنف (مثلث) والسبب في إعادة تصريحها هو أننا نرغب في تجاوزها بعد ذلك في الأسطر 10-16 قمنا بكتابة تعريف الدالة حتى تستطيع رسم شكل المثلث.

ملاحظة مهمة: حينما تقوم بتجاوز إحدى دالات الصنف الأب ، فإنك لا تتجاوز فقط الدالة نفسها وحسب بل تتجاوز أيضاً التحميل الزائد لتلك الدوال فلو افترضنا أنك في الصنف الأب قمت بزيادة تحميل الدالة Draw( ) لتصبح هكذا Draw (int) ، ثم قمت بتجاوز الدالة Draw( ) في الصنف الابن فكأنك في الحقيقة أخفيت الدالة Draw(int) الموجودة لدى الأب عن الصنف الابن .. وسيبلغ المترجم عن خطأ إذا قمت بإستدعائها في الصنف الابن ، وينبغي عليه تعريفها.

كيف نستفيد من الوراثة لأقصى حد ممكن:

قد تقول أننا حينما نجد عدة أصناف تشترك في عدد من الخصائص فإننا نقوم بصنع صنف أساس ثم نشتق منه هذه الأصناف ، هذا ليس خطأ من ناحية برمجية ولكنه خطأ كبير من ناحية التصميم ومن ناحية مبادئ البرمجة الشيئية ، فالبرمجة الشيئية أتت كمحاولة لتمثيل العالم الواقعي وبالتالي فأنت لا تقوم بجمع مجموعة متشابهة من الأصناف وصنع صنف أب ثم إشتقاق بقية الأصناف ، الفائدة الوحيدة لهذا العمل هو أنك قمت بتوفير مزيد من العمل في كتابة الكود ؛ لأقصى إستفادة ممكنة من البرمجة الشيئية فلا بد علينا من تمثيل العالم الواقعي في برامجنا. كما ترى في المثال السابق (وإن كان فيه أخطاء من ناحية التصميم) فلقد قمنا بكتابة الصنف الأب(الشكل أو ربما المستطيل) والصنف الابن (المربع) والصنف الحفيد (المثلث) وقمنا بإشتقاقها من بعضها وليس السبب هو وجود تشابهات بينهم بل لأنها في العالم الحقيقي هكذا ، فلو كان الذي نريده هو ليس هكذا لما أعدنا تعريف الدالة Draw( ) ، صحيح أن الصنف الحفيد يختلف عن بقية الصنفين في هذه الدالة ، إلا أنه بالفعل الصنف الأب والابن يمتلكان هذه الدالة ويختلفان فيها عن الصنف الحفيد بطريقة الإستخدام إلا أنهم يشتركون جميعهم في وجود هذه الدالة Draw ( ) ، وبالتالي فعلينا ألا ننسى مبدأ التجريد هنا عند تصميم أي أصناف تعتمد على الوراثة ، حتى تفهم المكتوب في الأعلى فربما عليك الانتظار قليلاً حتى نصل إلى موضوع الواجهات. لاحظ أيضاً أن مثال الأشكال تتحقق فيه بعض من الأشياء التي قلتها فالصنف الأب يمكن اعتباره مستطيل والصنف الابن المربع هو حالة خاصة من المستطيل والمثلث القائم الزاوية (الذي هو الصنف الحفيد) عبارة عن نصف مربع.

طريقة إستدعاء الدالة المتجاوزة في الصنف المشتق:

تستطيع إستدعاء الدالة المتجاوزة حينما تعمل على الصنف الابن ، سنقوم الآن بجعل الصنف الحفيد (المثلث) يقوم بإستدعاء دالة Draw الموجودة لدى الصنف الأساس shape ، انظر الآن إلى الدالة main() وكيفية فعل ذلك :

CODE

```
1. void main()
```

```
2. {int x;
3. do
4. {
5. cin >> x ;
6. cout << "\n";
7. triangle A(x);
8. A.shape::Draw();
9. } while (x!=0) ;
10.    }
```

كما تلاحظ فإن السطر الثامن هو الذي يقوم بإستدعاء الدالة Draw الموجودة لدى الصنف الأساس ، وكما ترى فإنه من الممكن إستدعاء الدالة Draw الموجودة لدى الصنف square بنفس الطريقة.

قاعدة:

لتتمكن من إستدعاء إحدى دوال الصنف الأساس لدالة تم تجاوزها في الصنف الابن فعليك إستدعاؤها بهذه الطريقة:

اسم الكائن المشتق	نقطة	اسم الصنف الأساس	معامل تحديد المدى	دالة الصنف الأساس
11. A	.	shape	::	Draw();

لقد انتهينا الآن تقريباً من مبادئ الوراثة والأساسيات الواجب توافرها لكي تتقدم أكثر في البرمجة الكائنية ، وسنتعمق الآن ونغوص أكثر في مبدأ التجريد بشكل خاص ومبادئ البرمجة الشيئية بشكل عام.

الدالات الظاهرية(الإفتراضية) Virtual Function:

للدالات الظاهرية فائدة كبيرة نوعاً ما حينما تتعامل مع مؤشرات لأصناف ، ولكن هذا الكتاب لن يقدم لك فائدتها البرمجية فحسب بل سيقدم لك فائدتها على مستوى الصعيد الكائني ، فتعلم الدالات الظاهرية سيزيد من مقدرتك على التجريد ومقدرتك أيضاً على صنع الواجهات والتعامل معها.
كما هو واضح من معنى الدالات الظاهرية فهي تعني أنها موجودة داخل تركيب صنف ما ، لكنها ليست موجودة في الواقع (على الصعيد البرمجي أقصد) ، قد تتساءل عن فائدتها إذاً؟ ، في الحقيقة فإن للدالات الظاهرية فوائد كثيرة ،سنأتي الآن بأحد فوائدها قم بدراسة المثال التالي:

CODE

```
1. #include <iostream.h>
2.
3. class Bird
4. {
5. public:
6.    Bird():itsAge(1) { cout << "Bird Alive...\n"; }
7.    ~Bird() { cout << "Bird die...\n"; }
8.    void fly() const { cout << "Bird fly away\n"; }
```

```
9.     void trills() const { cout << "Bird trills!\n"; }
10.   protected:
11.         int itsAge;
12.
13.   };
14.
15.   class Dicky : public Bird
16.   {
17.   public:
18.         Dicky() { cout << "Dicky Alive...\n"; }
19.         ~Dicky() { cout << "Dicky die...\n"; }
20.         void trills()const { cout << "oooooooooooo!\n"; }
21.         void fly()const { cout << "Dicky speed to...\n"; }
22.   };
23.
24.   int main()
25.   {
26.
27.         Bird *pDicky = new Dicky;
28.         pDicky->fly();
29.         pDicky->trills();
30.
31.         return 0;
32.   }
```

قمنا بإنشاء صنفين اثنين هـما Bird و Dicky ، ولا أعتقد أنك في حـاجـة لشرح تعريفات الدوال (لاعتقادي أنك وصلت مرحلة تمكنك من فهـمها) . كما ترى في السطر 27 قمنا بالإعلان عـن مؤشر يشير إلى كائن من الصنف Bird وحجزنا له ذاكرة من الصنف Dicky ؛ وبالطبع فإن السي بلس بلس تسمح بذلك لأن إستخـدام المؤشرات بهذه الطريقـة يعتبر آمناً ، قمنا الآن بإستدعاء دالتين في السطرين 28 و 29 ، فلنرى الآن إلى ناتج المثال السابق:

CODE

```
1. Bird Alive...
2. Dicky Alive...
3. Bird fly away
4. Bird trills!
```

كما ترى فإن ناتج البرنامج كان من المفترض ألا يكون هـكذا، لأنـك حجـزت لـه ذاكرة من النـوع dicky وليس من النـوع Bird ، وكـان الأحـرى أن يكـون نـاتج البرنامج هـكذا:

CODE

```
1. Bird Alive...
2. Dicky Alive...
3. Dicky speed to...
4. oooooooooooo!
```

والسبب في عـدم ظهـور هذا الناتج هـو أن المترجـم لا يعلم أي صنف يشير إليه pDicky حينما يتم تنفيذ البرنامج فعلياً ، لذلك فإن المترجم يقوم بإستدعاء الدالتين fly() و trills ( ) المعرفتين في الصنف الأساس بإعتبار أن الكائن pDicky يشير إلى الصنف الأساس Bird .
ولحل هذه المشكلة فعليك إخبار المترجـم أي دالة يستدعـي وحتى تنجح في ذلك فعليك جعل الدالات trills و fly دالات ظاهريـة أو إفتراضيـة وتعيد كتابة البرنامج ليصبح هـكذا بعـد التعـديل:

CODE

```
1. #include <iostream.h>
2.
3. class Bird
4. {
5. public:
6.    Bird():itsAge(1) { cout << "Bird Alive...\n"; }
7.    virtual ~Bird() { cout << "Bird die...\n"; }
8.    virtual void fly() const { cout << "Bird fly away\n"; }
9.    virtual void trills() const { cout << "Bird trills!\n"; }
10.   protected:
11.         int itsAge;
12.
13.   };
14.
15.   class Dicky : public Bird
16.   {
17.   public:
18.         Dicky() { cout << "Dicky Alive...\n"; }
19.         virtual ~Dicky() { cout << "Dicky die...\n"; }
20.         void trills()const { cout << "oooooooooooo!\n"; }
21.         void fly()const { cout << "Dicky speed to...\n"; }
22.   };
23.
24.   int main()
25.   {
26.
27.         Bird *pDicky = new Dicky;
```

```
28.         pDicky->fly();
29.         pDicky->trills();
30.
31.         return 0;
32.   }
```

كما ترى فلقد غيرنا تصريح الدالتين trilld و fly وجعلناها مسبوقة بالكلمة المفتاحية virtual ، هذا سيجعل البرنامج يستدعـي الدالتان الصحيحتـان وليس الدالتان في الصنف الأساس.

معلومة:

حينما تقوم بكتابة **virtual** قبل اسم أي دالة ضمن تركيب صنف ما ، فإنك تخبر المستخـدم أنـه سيتم تجـاوز هذه الدالة في الصنف المشتق من الصنف الأساسي ، بالتالي فإي حال ما قمنا بكتابة السطر التالي، ولم نقم بكتابة الكلمة **virtual**:

```
33.   Bird *pDicky = new Dicky;
```

فإن المترجم سيفترض أن المستدعـي يريد إستدعاء الدالة الموجودة في الصنف المتوفرة لديـه ، وكما ترى فإن الصنف يشير إلى الصنف الأساس والذي هـو **Bird** فإن المترجم لن يقوم بإعداد مؤشر الدالة ليشير إلى أعمق صنف مشتق قام بتجـاوزها بل سيشير إلى دالة الصنف الذي يشير إليه المؤشر أساساً إليه ، أما إذا قمت بكتـابة الدالـة الكلمة **virtual** فإنك تخبر المترجم أننا سنقوم بتجـاوز هذه الدالة في الصنف المشتق وبالتالي يعد مؤشر الدالة ليشير إلى المكـان الصحيح. بصراحـة فإن المترجم حينما تكتب له أمر مثل السطر السابق وقمت بإستدعاء إحدى الدوال فإنـه لن يدري أي دالة يستدعـي وسيترك الأمر لحين بدء التنفيذ وحينما تكتب كلمـة **virtual** فسيعلم أنك تقصد أعمق دالة ، أي أنـه سينفذ دالة الصنف المشتق وليس الأساس

كما ترى فإن الدالات الظاهريـة لن تعـمل إلا مع المؤشرات والمرجعيات ، ونصيحتـي لك هـي أن تبتعـد قدر الإمكـان عـن المؤشرات لأقصـى حـد ممكن وألا تستخدمها إلا في حالات معـينـة فقط تحتاج إليها بالفعـل.

التوارث المتعـدد :

لنفرض أنك تقوم بكتـابة مجمـوعـة أصناف لإستخـدامها لاحقاً في نظام ستنشئـه للجامعة ، هذه المجمـوعـة التي تكتبها هـي مجمـوعـة الأشخاص المنتمين للجامعة ، فإن أول ما تفكر بـه هـو إنشاء صنف شخص ثم تتشق من هذا الصنف الأساسـي صنفي الإداري والدكـتور ، لكن إذا وصلت لإنشاء صنف مدير القسـم ، فستتساءل عـما ستقوم بإنشاءه هل تشتق هذا الصنف من الإداري أم الدكتـور ، في الحقيقـة فإن السـي بلس بلس توفر لك إمكـانية أن تشتق الصنف مدير القسم من الصنفين الاثنين (أي الإداري والدكتـور) وهذا ما يعرف بالتوارث المتعـدد ، سنقوم بكتـابة أحـد الامثلة التوضيحية هـاهـنا:

CODE

```
1. #include <iostream.h>
2.
3. class Employee
4. {
5.    protected:
6.    int itsAge;
```

```
7. public:
8.    Employee():itsAge(0){cout << "\nHii I am Employee\n";}
9.    Employee(int  x):itsAge(x)  {cout  <<"\nHii  I  am  Employee(int)
   \n";}
10.         Getme(){cout <<"\n Hiii  I am I am Employee\n";}
11.   };
12.
13.   class prof
14.   {protected:
15.         int itsAge1;
16.   public:
17.         prof():itsAge1(0){cout << "\nHii I am prof\n";}
18.         prof(int  x):itsAge1(x)  {cout  <<"\nHii  I  am  prof(int)
   \n";}
19.         Getme(){cout <<"\n Hiii  I am I am prof\n";}
20.   };
21.
22.   class chief:public prof,public Employee
23.   {public:
24.         chief(){cout << "\nHii I am chief\n";}
25.         chief(int x) {cout <<"\nHii I am chief(int) \n";}
26.   };
27.
28.   void main()
29.   {
30.         chief ml(9);
31.         ml.prof::Getme();
32.   }
```

وهذا هو ناتج البرنامج:

Hii I am a proof

Hii I am Employee

Hii I am chief(int)

Hii I am I am chief

كما ترى فلقد قمنا بالتصريح عن صنفين اثنين الأول هو Employee والثاني هو prof في السطرين 3 و13 ؛ ثم قمنا بإنشاء الصنف chief في السطر 22، وهذا الصنف الجديد يرث من صنفين اثنين وليس من واحد فقط كما تعودنا

خـلال الأمثلـة السابقـة ؛ جميـع دوال البنـاء فـي الـثلاث أصـناف تطبـع جملـة واحدة تدل على إنشـاءها وهــي بـالطبع لهـا دالتـي بنــاء إحـداها دالـة البنـاء الإفتراضية والأخـرى دالة البناء تأخذ عـدد معين كوسـيط لهـا ؛ يـتم التصـريح عـن التوارث المتعـدد كما في السطر 22:

قاعدة:
الإعلان عن التوارث المتعدد يتم عن طريق الفصل بين الأصناف المشتقة بواسطة فاصلة (,) ولا يشترط الإستقاق من صنفين بل يجوز الإشتقاق من أكثر من صنفين:

الصنف المشتق	نقطتين	الصنف الأب الثاني	فاصلة	اسم الصنف الأب الاول
class chief	:	public prof	,	public Employee

دوال البنـاء والهـدم في التوارث المتعـدد :

في المثال السـابق وحسـب مـا هـو موجـود فـي السـطر 22 فلقـد منـا أولاً بإشتقاق الصنف prof ثم قمنا بإشتقاق الصنف Employee بالتالي فإن دوال البناء التي ستظهر أولاً هـي حسبما تطلبـه الصـنف chief ، وكمـا تـرى فـي السطر 30 فلقد أعلنا عـن كائن اسمـهml ومررنا له عـدد صحيح بالتـالي فـإن المترجم سيستدعـي دالة البناء الخاصـة بـ chief والتي تستقبل عـدد صحيح وكما ترى مـن تعريف الدالـة فـي السـطر 25 فهــي لـم تطلـب مـن المتـرجم إستدعاء دوال بناء الصنفين الآخرين بل تركت الأمر له حتـى يفعـل مـا يريـده ، بالتالي فإن المترجم سيقوم بإستدعاء دالـة الصـنف المشـتق الأول ثـم دالـة الصــنف المشــتق الثــاني أي أنــه ســيقوم بإســتدعاء الدالــي prof ثــم دالــة Employee. تستطيع أن تطلب من المترجم أن يترك هذه الطريقـة الإفتراضية ويستدعـي دالة بناء الصنف prof التي تتمكن من تمرير عــدد وسـيط لهـا ثـم دالـة بنـاء الصـنف Employee الإفتراضيــة لكنــك لـن تسـتطيع تغييـر ترتيـب إستدعاء دوال البناء .

الدوال الاخرى وكيفيـة إسـتدعاؤها :

هل ترى إستدعاء الدالـة ( ) Getme فـي السـطر 31 لـو كـان هـذا الإسـتدعاء مكتـوباً بهذه الطريقـة:

```
33.          ml.Getme();
```

لما أستطاع المترجم أي دال تقصد ، فهل هي الدالـة التـي قمـت بإشـتقاقها من الصنف Employee أم من الصنف prof ، فالصنفين جميعهما يملكـان هـذه الدالة ، وبسبب ذلك فـإن المتـرجم سـيختلط عليـه الأمـر ولـن يعـرف أي دالـة تقصد ، أما إذا قمت بتجــاوز الدالـة ( ) Getme فلـن يكـون هــناك أي مشـكلـة في الأمـر ، أما وفي حال لم تقم بتجـاوزها فعليك أن تحـدد للمتـرجم أي دالـة تقصد وبسبب ذلك فبإمكـانك تعـديل السطر 33 ليستدعـي الدالـة ( ) Getme الخاصـة بالصنف Employee كما هـو واضح في هذا السطر:

```
34.          ml.Employee::Getme();
```

والامر في الحقيقـة يشبـه ما تكلمنا عـنه من طريقــة إسـتدعاء دالـة الصـنف الأساس من الصنف المشتق في حال تم تجـاوز الدالة المعــنية فـي الصـنف المشـتق.

سنقوم بالتعـديل في المثال السابق ، وسنطبق مبادئ البرمجـة الكائنية في هذا التعـديل حتى وإن كان طفيفاً:

CODE

```
1. class person
2. {
3. public:
4.    person():itsAge(0){cout << "\nHii I am Person\n"; }
5.    person(int x):itsAge(x){ cout <<"\nHii I am Person (int)\n";}
6.     Get() { cout << "\nGettttttttttttttt\n";}
7. protected:
8.    int itsAge;
9. };
10.   class Employee: public person
11.   {
12.   public:
13.         Employee(){cout << "\nHii I am Employee\n";}
14.         Employee(int   x):person(x)   {cout   <<"\nHii   I   am
   Employee(int) \n";}
15.         Getme(){cout <<"\n Hiii  I am I am Employee\n";}
16.   };
17.
18.   class prof: public person
19.   {
20.   public:
21.         prof(){cout << "\nHii I am prof\n";}
22.         prof(int x):person(x) {cout <<"\nHii I am prof(int) \n";}
23.         Getme(){cout <<"\nHiii  I am I am prof\n";}
24.   };
25.
26.   class chief:public prof,public Employee
27.   {public:
28.         chief(){cout << "\nHii I am chief\n";}
29.         chief(int  x):Employee(),prof(x)  {cout  <<"\nHii  I  am
   chief(int) \n";}
30.   };
```

لم نقم في الكود السابق بالكثير بل فقط كل الذي قمنا به هو أننا قمنا بإضافة صنف اسمه person **قمنا بإشتقاق الصنفين** Employee **و** prof **منه.**
بالرغم من أننا ستعتقد بديهياً أن الصنفين Employee **و** prof **قد ورثا أغلب أعضائهما من الصنف الأساس** person **إلا أننا إذا نظرنا من رؤية صحيحة فهما في الحقيقة لم يرثا من نسخة واحدة من الصنف** person **بل كل صنف منهما تورث صفاته من نسخة أخرى مختلفة عن النسخة الأساس للآخر، وحتى نوضح أكثر ما أقصد ، فأنت ترى في السطر** 6 **دالة جديدة أسمها** Get( ) **هذه الدالة لم نقم بتجاوزها في الصنفين** Employee **و** prof **وبالطبع**

فإن هـذان الصنفـان لهـما نسـخـة مختلفـة عـن الأخـرى بالنسـبـة للدالة Get ( ) وعـنـدما تقوم بتوريث هذه الدالة إلى الصنف chief فإن هذا الصنف سيملك نسختين من الدالة Get( ) بالرغـم من أن مصدر هذه الدالة واحد ألا وهـو الصنف person وبالتالي فعـنـدما تكتب هذا المثال فإن السطر الثاني خاطئ:

CODE

```
1.   chief ml(9);ml.prof::Getme();
2.    ml.Get();
```

السطر الاول يعلـن عــن كــائن مـن الصـنف chief أما التعليمـة الثانيـة فهــي تستدعــي الدالة Getme( ) الموجودة في الصنف prof ، أما بالنسبة للسـطر الثـاني فهــو يستدعــي Get( ) الخاصـة بالصـنف person ونظـراً لأنــه يوجـد نسـختين اثنتـين مـن الصـنف person ، فـإن المتـرجم سـيخلط بـين أي دالـة تريدهـا.
هـناك أحد الحـلول لهذه المشـكلـة ألا وهـو تجاوز الدالة Get( ) في الصنف chief فبإمكـانك كتـابة السـطور التاليـة:

CODE

```
1.   int chief::Get()
2. {
3.    return prof::Get();
4. }
```

لم نفعل الكثير سوى أننـا فـي السـطر الثالـث قمنـا بإسـتدعاء الدالـة Get( ) الخاصــة بالصـنف prof . وبالتـالي فـإن السـطر الثـاني مـن المثـال السـابق سيعـمل دون أية مشاكل. لكن مـاذا لـو أردت لأي سـبب مـن الأسـباب الدالـة Get الموجودة في الصنف person لأي سـبب مـن الأسـباب، فمـاذا عليـك أن تفعـل؟. أحـد الحـلـول هــو أن تجعـل الصـنفين Employee و prof أن يرثـا مـن نسـخـة واحدة من الصنف person وليس نسختين كمـا هــو الحـال. وسـبيلك لفعل ذلك هـو الوراثـة الظاهرية (التوارث الظاهري).

الوراثة الظاهريـة:

حينما تقوم بجعل الصنفين Employee و prof يرثان الصنف person فإنهـما في الحقيقـة يرثـان من نسـخـة واحدة من الصنف ، ولن يرثان من نسختـان اثنـان من الصنف person لذلك فهـو يختلف عـما عليه الحال في التوارث المتعـدد الغير ظاهري. ونظراً لوجود نسخـة واحدة من الصنف person فإن الصنف chief بإمكـانـه تهيئتها حسبما يريد دون أن يهـتم بكيفيـة تهيئـة الكائنين الآخرين للصنف person.انظر لهذا الكـود وهـو نفس الكـود السابق مع بعض التعـديلات:

CODE

```
1.   #include <iostream.h>
2.
3.
4. class person
5. {
```

```
6. public:
7.    person():itsAge(0){cout << "\nHii I am Person\n"; }
8.    person(int x):itsAge(x){ cout <<"\nHii I am Person (int)\n";}
9.    int Get() { cout << "\nGettttttttttttttt\n";return itsAge;}
10.         GetItsAge(){return itsAge;}
11.   protected:
12.         int itsAge;
13.   };
14.   class Employee: virtual public person
15.   {
16.   public:
17.         Employee(){cout << "\nHii I am Employee\n";}
18.         Employee(int   x):person(x+2)   {cout   <<"\nHii   I   am
   Employee(int) \n";}
19.         Getme(){cout <<"\n Hiii  I am I am Employee\n";}
20.   };
21.
22.   class prof: virtual public person
23.   {
24.   public:
25.         prof(){cout << "\nHii I am prof\n";}
26.         prof(int  x):person(x+2)  {cout  <<"\nHii  I  am  prof(int)
   \n";}
27.         Getme(){cout <<"\nHiii  I am I am prof\n";}
28.
29.   };
30.
31.   class chief:public prof,public Employee
32.   {public:
33.         chief(){cout << "\nHii I am chief\n";}
34.         chief(int x):person(x*2) {cout <<"\nHii I am chief(int)
   \n";}
35.         Get();
36.   };
37.
38.   int chief::Get()
39.   {
40.         return prof::Get();
```

```
41.   }
42.   void main()
43.   {
44.         chief ml(10);
45.         ml.person::Get();
46.         cout << ml.GetItsAge()<< endl;
47.   }
```

أهم التغيرات الواقعة هي السطرين 14 و 22 حيث قمنا بالإعلان أن الصنفين Employee و prof سيرثان ظاهرياً من الصنف person وكما ترى في السطرين 18 و 26 غيرنا إعدادات البناء الخاصة بالصنف person فالصنفين الآن (Employee و prof) يقومان بزيادة المتغير itsAge بالعدد 2 ، أما بالنسبة للسطر 34 فلقد أصبحت الصنف chief تغير من إعدادات البناء الخاصة بـ person ، فهي الآن تضاعف العدد مرتين . هذا يعني أن الصنف chief تجاوز دالتي البناء في Employee و prof وهذا أحد الإختلافات في التوارث الظاهري عن غيره من التوارث.

أما بالنسبة لفوائد الوراثة الظاهرية فهي ما أنت تريده بالفعل في برنامجك إذا افترضنا أن مستويات الوارثة وصلت لديك إلى المستوى الخامس (أي الجيل الخامس) فربما ترغب بأن تكون أصغر صنف مشتق قادر على تعديل صفات نسخة الصنف الأساس الذي يقوم بالإشتقاق منه.

الأصناف المجردة :

سنرجع الآن إلى أول مثال في وحدة الوراثة ألا وهو الأصناف التي تقوم برسم الأشكال الهندسية ، كما تلاحظ في الصنف shape فإنه لن يكون بإمكانك إشتقاق أو على الأقل نظرياً صنف الدائرة إلا إذا قمت بتجاوز الكثير الكثير من الدوال وكذلك الحال بالنسبة للمعين أو إذا رغبت برسم شكل بيضاوي ، الحل الوحيد هو أن تجعل الصتف الأساس shape صنفاً مجرداً من البيانات ، أي لا يحتوي على أي شيء ، كل الذي يحتويه هو أسماء الدوال والبيانات ، الغرض من هذه الأسماء هو وضع خطوط إرشادية لمن يريد الإستقاق من هذا الصنف ، وبصراحة فإذا نظرنا للواقع فإنه لا يوجد شيء اسمه الصنف shape ، إذا لفظت هذه الكلمة shape فإنها تدل على جسم ذو خصائص عامة لا يختلف فيها عن البقية فلربما يكون مربعاً أو دائرة أو ربما حتى خريطة أو صورة لشخص ، لذلك فمن الأفضل أن تجعل الصنف shape صنفاً مجرداً أو ما يلفظ باللغة الإنجليزية ADT إختصاراً لكلمة Abstract Dat Type أي نوع مجرد من البيانات.
النوع المجرد من البيانات ليس له وجود في الواقع إنما هو في الحقيقة مفهوم أو فكرة للأصناف الأخرى التي تشابهه.

الدالات الظاهرية الخالصة:

تدعم لغة السي بلس بلس النوع المجرد من البيانات عن طريق إنشاء صنف مجرد والطريقة لفعل ذلك ، هي أن يحتوي الصنف الذي ترغب بتجريده على دالة ظاهرية خالصة ولو كانت واحدة ، الصنف المجرد لا يمكنك إنشاء كائنات منه بل هو فقط مخصص للأصناف المشتقة صحيح أنه بإمكانك تعريف دوال الصنف المجرد إلا أن ذلك فقط لزيادة النواحي

الوظيفـية أو الإجرائيـة للصنف المشتق وبإمكـانك بعـد إذا إستدعاء هذه الدوال من الصنف المشتق ، سنقوم الآن بإنشاء مجموعـة الأصناف التي ترسم الأشكال الهـندسيـة ، لكن تذكر أننا نكتبها لأغراض تعليمية وليس من أجل أن نقوم بالفعل برسمها ، إذا انتهيت من قراءة هذه الوحـدة فبإمكـانك مراجعة قسم الأكـواد لمعرفـة كيف نستطيع تطبيق هذه المبادئ على أرض الحقيقـة ، لا تحـاول الذهاب الآن ، حاول أن تصبر حتى تفهـم هذه المبادئ أولاً :

CODE

```
1.   class shape
2. {
3. protected:
4.    int d1,d2,d3,d4;
5.
6. public:
7.    shape() {d1=d2=d3=d4=5;
8.    cout << "\nHere I am shape ( ) \n" ;}
9.    shape (int x,int y)
10.         {d1=d3=x;d2=d4=y;
11.         cout << "\nHere I am SHAPE (INT,INT)\n";
12.         }
13.
14.         shape (int a,int b,int c,int d)
15.         {
16.               d1=a;d2=b;d3=c;d4=d;
17.               cout << "\nHere I am SHAPE (INT , INT , INT
   ,INT)\n";
18.         }
19.
20.         virtual void draw() =0;
21.         virtual ~shape(){ cout <<"\n I am diee (shape)\n " ;}
22.   };
23.
24.   class Rectangle:public shape
25.   {
26.   public:
27.         Rectangle(){ cout<<"\nHere I am Rectangle() \n";}
28.         Rectangle(int a,int b):shape(a,b)
29.         {cout << "\nHere I am Rectangle (INT  , INT) \n";}
30.         Rectangle(int a,int b,int c,int d):shape(a,b,c,d)
31.         { cout <<"\nHere I am Rectangle (INT,INT,INT,INT)\n";}
```

```
32.         void draw() {
33.         for (int i=0;i<d1 ;i++)
34.         {
35.         for (int j=0;j<d2;j++)
36.         cout << "*";
37.         cout <<endl;
38.         }
39.         }
40.         ~Rectangle(){cout<< "\n I am diee (Rectangle)\n";}
41.
42.   };
43.
44.   class Circle:public shape
45.   {
46.   public:
47.         Circle(int m){cout << "Here I am Circle (INT) ";
48.         itsd=d1;}
49.         void draw() { cout <<"\n Here I am draw a Circle has  ";
50.         cout << itsd << "  cm \n";}
51.         ~Circle(){ cout <<"\nI am die Circl \n";}
52.   private:
53.         int itsd;
54.   };
```

كما ترى فلقد قمنا بإنشاء ثلاثة أصناف هم الصنف شكل shape والصنف مستطيل Rectangle والصنف دائرة Circle ، ولقد جعلنا لكل صنف دالة بناء تقوم بطباعة جملة حينما نقوم بإنشاءها حتى تتأكد بالفعل أنها أنشئت ودوال هدم تطبع رسالة تفيد أنها هدمت ، المهم في الأمر أنه في السطر 20 قمنا بكتابة دالة رسم تابعة للصنف shape طريقة التصريح الغريبة عن هذه الدالة تفيد بأنها دالة ظاهرية خالصة وبالتالي فالصنف shape هو صنف مجرد لا يمكنك إنشاء كائن منه ، حينما تقوم بالتصريح عن دالة ظاهرية خالصة فأنك تخبر المترجم بما يلي:

- أن الصنف الذي يحتوي هذه الدالة هو صنف مجرد ADT.
- أنه يجب على بقية الأصناف التي تشتق من هذا الصنف المجرد أن تقوم بتجاوز الدالة الظاهرية الخالصة في الصنف الأساس حتى وإن كان جسم الدالة الظاهرية الخالصة مكتوباً ، وفي حال عدم تجاوز هذه الأصناف لهذه الدوال فإنها تعتبر نوع مجرد ADT.

بامكانك أيضاً كتابة جسم الدالة الظاهرية الخالصة ضمن تعريف الصنف ADT ولكن حتى مع ذلك فيجب عليك تجاوزها في الأصناف المشتقة.

إلى هـنا انتهينـا مـن وحـدة الوراثــة وتعــدد الأوجــه ، ولـم نتعـرض فيهـا إلـى تطبيقات عـملية بل تركـنا ذلك إلى الصفحـات الأخـرى.

القوائم المترابطة

Linked List

بدايـة:

القوائم المترابطة إحدى الفوائد الكبيـرة التـي أتـت بهـا البرمجـة الكائنيـة أو بالأصح قامـت بتعـزيزها بمفهـوم الكائنـات وإن كـانت موجـودة فـي اللغـات الإجرائيـة إلا أننا نراها هـنا في البرمجـة الكائنيـة بشكل أفضل.

مدخل:

تعرضـنا فـي وحـدة سابقـة (وحـدة المؤشـرات) علـى مفهـوم المصفوفـة الديناميكيـة والتي يستطيع المستخـدم تغييرها متى ما أراد ، وقد جعلنـا مـن المصفوفـة مرنـة بشكل كبير ، فأصبح المستخـدم هـو الـذي يحـدد حجمهـا ، إلا أن هـنـاك بعـض المشـاكل حتـى مـع المصفـوفـة الديناميكيـة الجـديدة ، فلنفرض مثلاً أننا نطـور نظامـاً لإدارة المكتبـات العامـة ، وبالتحـديد لتسـجيل الكتب في المكتبـة ، ولقـد طلـب منـك أمـين المكتبـة أن يكـون عـدد الكتـب المسموح بتسجيلها في النظام 1000 كتـاب ، ولكن بعـد شـهرين أتـى إليـك وشكـى بأن البرنامج لم يسجل الدفعـة الجـديدة مـن الكتـب ، والسـبب فـي ذلك أن نظامك لا يستطيع إستقبال أكثر من ألف كتـاب ، ألم يكن من الافضل أن تقـل لصـاحب المكتبـة بأنـه هـو الـذي يحـدد عـدد الكتـب المسمـوح بتسجيلها داخل النظام سواءً أراد 1000 كتـاب أم 2000 كتـاب ، أيضـاً فلننظر إلى الأمر من ناحية نظام التشـغيل ، ألـن يكـون هـنـاك مشـاكل كبيـرة حينمـا تطلب من البرنامج حجز أكثر من 1000 عـنصر دفعـة واحدة، ماذا لو فكرت بأن مستخـدم النظام حينما يقوم بتسجيل كتـاب يقوم البرنـامج بتخصيص ذاكـرة محددة له ثم بعـد ذلك إذا أراد التسجيل مرة أخــرى يقـوم الكتـاب بتسـجيلها مـرة أخــرى ، هـذه المشـاكل الكبيـرة نسـبياً لا تجعـل مـن أمـر المصفوفـة الديناميكيـة أمراً رائعـاً ، ألا توافقـني في الرأي.

سلسلـة من المؤشرات:

دعـنا ننظر الآن إلى المصفـوفـة ، أنت تعلم بأن المصفـوفـة عبارة عـن بيـانات متجـاورة مع بعضها البعض ، بالتالي فحينما ينتقـل البرنـامج مـن عـنصر إلـى آخر فهـو فـي الحقيقـة يزيـد عـدد مـن البايتـات علـى موقـع ذاكـرة العـنصر لينتقل إلى العـنصر الآخر ، والزيادة هذه حسـب نـوع المصفوفـة أهـي char أم int ، وتسـتطيع التأكـد مـن هـذه النقطـة عـن طريـق طبـاعـة عـناوين عـناصر المصفوفـة (مـن النـوع int مـثلاً) وسـترى أن الفـرق بـين كـل عـنصر وعـنصر هـو 2 أو 4 ، وبالتالي فيمكن تشبيـه المصفـوفـة علـى أنهـا لـوح مـن الشطرنج مقسم إلى مربعـات ، لكن ما رأيك لو نـطور مصفوفة أخـرى وهـذه المرة لن ننظر إليها على أنها لوح من الشطرنج بل علـى أنهـا مجموعـة مـن الأفراد الذين يشـيرون إلـى بعضهـم ، فمـثلاً لـو ذهـبنا إلـى أول عـنصر فـي نمـوذج المصفـوفـة المقترحـة (ولنفتـرض أنهـا مصفوفـة أعـداد) فلـن نجـد العـنصر التالي بجانبـه بل سيخبرك أنـه يستطيع نقلك إلى عـنصرين اثنـين أو حتى ثلاثـة ، سيقول لك إذا كـنت تبحث عــن عـدد أقـل مـن 40 فاذهـب إلـى هذا العـنصر واستمر في البحث ، وإذا أردت عـدداً أكبـر مـن 40 فاذهـب إلـى العـنصر الآخر ، وهذا العـنصر الأول ليسـت العـناصر الأخـرى بجانبـه بـل هـو يحوي عـناوينهـا ، أي أن الأمر عبارة عـن سلسلـة من المؤشرات التـي تشـير

إلى بعضها ، فالعنصر الأول يشير إلى مجموعة من العناصر التي تحوي أعداد أكبر من 40 ويشير أيضاً إلى مجموعة العناصر التي تحوي أعداد أقل من 40 (ولنفرض أنك تريد العنصر 20) بالتالي فأنت ستذهب إلى العنصر الثاني وحينما تصل إليه سيخبرك بأنه يشير إلى مجموعة العناصر التي تقل أعدادها عن 15 ويشير أيضاً إلى مجموعة العناصر التي تزيد أعدادها عن 15 ، فبالتالي ستذهب إلى المجموعة الثانية وهكذا دواليك حتى تصل إلى النقطة التي تريدها ، بالتالي فإن القائمة المترابطة سهلت علينا البحث جدياً ، بإختصار القائمة المترابطة هي عبارة عن سلسلة من المؤشرات التي تشير إلى العناصر التالية في سلسلتها ، الآن سنذهب إلى الصعيد الكودي وهذه الوحدة لن يكون لها قسم عملي أو كودي كما هو الحال مع الوحدات الأخرى نظراً لأهمية هذا الموضوع ولصعوبته النسبية عن باقي المواضيع فلقد تغير أسلوب الكتاب ليعطيك أمثلة عملية مباشرة دون الخوض في أمثلة توضيحية ليس لها أي مقصد فحسب رأيي فإن موضوع القوائم المترابطة يعتبر من أغمض المواضيع (وليس أصعبها) نظراً لأنه يعتمد على المؤشرات.

مثال/1

سنقوم في هذا المثال بكتابة نظام أو برنامج لإحدى الجامعات ، هذا البرنامج يقوم بتسجيل المقررات الدراسية ودرجتها النهائية وعدد ساعاتها ويمكن للمستخدم البحث في هذا البرنامج عن مقرر بعينه وطباعة بياناته أو حتى رؤية جميع بيانات المقررات المسجلة في النظام الجامعي.

الحل:

سنقوم بكتابة هذا البرنامج هكذا:

في البداية وكما تعلم فيجب علينا حل هذا المثال بواسطة القائمة المترابطة Linked List ، وليس بطريقة أخرى فلن يمكن حله بواسطة المؤشرات أو المصفوفات أو غيرها فالمصفوفات حجمها ثابت والمصفوفة الديناميكية يجب أن تكون ثابتة في إحدى نقاط تنفيذ البرنامج ولن يمكنك تغييرها بعد ذلك إلا بطرق غير عملية بتاتاً وتزيد من تعقيد البرنامج فقط.

أول بنية للصنف يجب تركيب البرنامج من خلالها هي صنف المادة الدراسية أو المقرر الجامعي والتي يجب أن تحتوي على مؤشر إلى المقرر الآخر من القائمة المترابطة.

بالتالي فإن تركيب الصنف أو التركيب سيكون هكذا:

CODE

```
49.   struct link
50.      {
51.         int number;
52.      float degree;
53.      int hours;
54.      link* next;
55.      };
```

في السطر الاول قمنا بالإعلان عن التركيب link وفي السطر الثالث احتوى التركيب على رقم المادة وفي السطر الرابع على درجتها النهائية

وفي السطر الخامس احتوى على عدد ساعات هذه المادة وفي السطر السادس والذي سيربط عناصر القائمة المترابطة مع بعضها البعض برباط وثيق قمنا بالتصريح عن التركيب التالي أو المادة التالية من القائمة المترابطة.

قمنا بتسمية التركيب الأساسي link بدلاً من course لأنه هو الذي يربط بين عناصر القائمة المترابطة ، وهذه التسمية ما أتت إلا لأغراض تعليمية وليس لأساس آخر وبالتالي فإذا رغبت في تطوير هذا التركيب فربما تغير اسمه إلى مسمى course.

هناك ملاحظة جديرة بالذكر إلا وهي أنه ليس بإمكانك أن تجعل التركيب السابق يحتوي على عنصر من نفس التركيب فمثلاً السطر التالي خاطئ مئة في المئة:

```
1. struct link
2. {
3. link  m;
4. }
```

لا يمكن للتركيب أو الصنف أن يحتوي على عنصر من نفس تركيبه أو صنفه أو نمطه إن شئت ، ولكن بإمكانه أن يحتوي على مؤشر من نفس النوع ، لأن هذا المؤشر لا يحجز ذاكرة في الأساس وإنما يشير إلى نوع بيانات آخر.

سنأتي الآن إلى إحدى النقاط الهامة جداً ألا وهي تركيب الصنف الآخر ، كيف سيكون شكله وكيف سينظم عمل البرنامج وكيف سندخل فيه مهام البحث وعرض المقررات الدراسية وما إلى ذلك من مهام النظام أو البرنامج الذي نقوم بصنعه حالياً.

الصنف الجديد هو عبارة عن القائمة المترابطة link list والذي يتحكم تحكماً تاماً بجميع عناصر التركيب link ، هذه هي بنية الصنف الجديدة:

CODE

```
1. class linklist
2.    {
3.    private:
4.       link* first;
5.    public:
6.       linklist()
7.       { first = NULL; }
8.       void additem(int d);
9.       void display();
10.           void find (int f);
11.           void Enter();
12.       };
```

يحتوي الصنف linklist على نؤشر خاص وحيد ألا وهو مؤشر إلى أول صنف في القائمة ، لم يحتوي التصريح السابق إلا على تعريف دالة البناء، حيث تقوم بجعل المؤشر يشير إلى لا شيء.

بالنسبة للسطر 8 فالدالة الموجودة به additem( ) تقوم بإنشاء مقرر جديد حسب الطلب .
بالنسبة للسطر 9 فهو يقوم بعرض جميع محتويات القائمة .
بالنسبة للسطر 10 فالدالة به find تقوم بإيجاد المادة أو المقرر الذي تريده .
بالنسبة للسطر 11 فيحوي الدالة Enter والتي تطلب من المستخدم إدخال جميع بيانات المقرر الجامعي ، سنأتي الآن إلى شرح جميع الدوال واحدة واحدة.

الدالة additem( ):

هذه الدالة أهم دالة موجودة في البرنامج حيث تقوم بإنشاء المقررات الجامعية وإضافتها إلى القائمة المترابطة ، وهذا هو تعريف هذه الدالة:

```
1. void linklist::additem(int d)
2. {
3. link* newlink = new link;
4. newlink->number = d;
5. newlink->next = first;
6. first = newlink;
7. }
```

في السطر الثالث قمنا بإنشاء مؤشر جديد وحجز ذاكرة له من النوع link في السطر الرابع قمنا بإسناد البارامتر الممرر للدالة إلى رقم المادة في المؤشر الجديد ، أما بالنسبة في السطر 5 فقمنا بإسناد المؤشر first إلى المؤشر next وهكذا فلقد أسندنا المؤشر first والذي لا يساوي أي شيء حسب دالة بناء الصنف إلى المؤشر new link ، حتى نفهم بشكل أفضل فدعنا نقوم بإختبار مالذي سيحدث إذا قام البرنامج بتنفيذ السطر التالي:

```
1. additem( 5);
```

في البداية سيتم إنشاء كائن من الصنف linklist هذا الكائن سيجعله المؤشر first يشير إلى لا شيء كما هو موضح في دالة بناء الصنف ، الآن ستقوم الدالة additem بإنشاء مؤشر من النوع link وستقوم بإسناد القيمة 5 إلى المتغير number في المؤشر الجديد ، وتقوم أيضاً بإسناد المؤشر first (الذي يشير إلى لا شيء) إلى المتغير next في المؤشر الجديد newlink ، الآن أصبح العنصر newlink في القائمة المترابطة يرتبط بلا شيء حسب السطر الخامس، الآن في السطر السادس يقوم البرنامج بأخذ عنوان المتغير أو المؤشر newlink وجعل المؤشر first يشير إليه ؛ وبالتالي فلقد أصبح المؤشرات newlink و first يشيران إلى نفس المنطقة من الذاكرة بعد ذلك ينتهي تنفيذ الدالة ويستكمل البرنامج عمله وهذه المرة قمنا بتنفيذ السطر التالي بعد السطر السابق مباشرة:

```
2. additem( 6);
```

سيتم الآن إعادة تنفيذ الدالة additem بنفس الطريقة إلا أن النتائج ستكون مختلفة.

في السطر الثالث تنشئ الدالة المؤشر newlink وهذه المرة يختلف عن المؤشر newlink الذي تم تنفيذه في السابق لأنه يحجز له مكان جديد في الذاكرة ، معروف ماذا يؤدي السطر الرابع ، بالنسبة للسطر الخامس فلقد تغيرت الفائدة منه ، الآن هل تعرف ما هو المؤشر first أو ما هي المنطقة التي يشير إليها ، إنها نفس المنطقة التي كان يشير إليها المؤشر newlink في المرة السابقة (حينما كان الرقم 5) يأخذ البرنامج المنطقة التي يشير إليها هذا المؤشر ويجعل المؤشر next (في التركيب newlink) يشير إلى نفس منطقة الذاكرة ، الآن أصبح المؤشر newlink الجديد الذي يحوي العدد 6 يحوي متغير يشير إلى المؤشر newlink القديم الذي يحوي العدد 5 ؛ الآن في السطر السادس نجعل المؤشر first يشير إلى نفس منطقة الذاكرة التي يشير إليها المؤشر newlink ، الآن هذه القائمة تحتوي على عنصرين سيقوم البرنامج الآن بإضافة عنصر جديد وثالث حتى تكتمل صورة الشرح ، انظر إلى السطر الجديد:

```
3. additem( 7);
```

سيتم تنفيذ هذا السطر بنفس الطريقة السابقة ، إلا أنه في السطر الخامس يقوم البرنامج بجعل المؤشر next في التركيب newlink يشير إلى نفس منطقة الذاكرة التي يشير إليها حالياً first والتي هي نفسها التي يشير إليها المؤشر newlink(6).

الآن هذه القائمة المترابطة تحوي ثلاثة عناصر الأول newlink(5) والثاني newlink(6) والثالث newlink(7) ، سنرى الآن كيف تتصل هذه العناصر بعضها ببعض.

الآن في المؤشر first يشير إلى نفس منطقة الذاكرة التي يشير إليها العنصر newlink(7) حسب آخر تنفيذ للدالة additem( ) الآن هذا العنصر يحتوي على مؤشر يشير إلى نفس منطقة الذاكرة التي يشير إليها العنصر newlink(6) ، هذا العنصر newlink(6) يحتوي على مؤشر وهو next يشير إلى نفس منطقة الذاكرة التي يشير إليها المؤشر newlink(5) بهذا الشكل ترتبط عناصر القائمة المترابطة بعضها ببعض فالعنصر الأول يشير إلى أحد العناصر وهذا العنصر يشير إلى عنصر آخر وهكذا دواليك حتى النهاية ، بالمناسبة هل تعرف ماهي المنطقة التي يشير إليها المؤشر next في العنصر newlink(5) ؟ ؛ ارجع إلى الأسطر السابقة حتى تعرف ما هي المنطقة التي يشير إليها ذلك المؤشر.

الدالة Enter( ):

لا جديد في هذه الدالة وهذا هو تعريفها:

```
1. void linklist::Enter()
2.  {
3.    cout << "Enter its Degree:";
4.    cin >> first->degree;
5.    cout << "Enter its hours: ";
6.    cin >> first->hours;
7.
8.
9.   }
```

الدالة Display( ):

هذه الدالة مهمة للغاية ؛ انظر إلى تعريف هذه الدالة:

```
1. void linklist::display()
2.    {
3.    link* temp = first;
4.    cout << "\n\n--------------------------------------------\n";
5.    while( temp != NULL )
6.       {
7.    cout <<"Number Of Course:\t" << temp->number << endl;
8.    cout << "its degree:\t\t" << temp->degree << endl;
9.    cout << "its Hours:\t\t " << temp->hours << endl;
10.   cout << "--------------------------------------------\n";
11.         temp = temp->next;
12.         }
13.      }
```

أول شيء يقوم به هذه الدالة هو إنشاء مؤشر مؤقت هو temp يشير إلى نفس منطقة الذاكرة للمؤشر first والذي هو نفسه المؤشر newlink(7) ؛ بعد ذلك يدخل البرنامج في الدوارة الشرطية while والتي من أهم شروطها ألا يشير المؤشر temp إلى الصفر أو القيمة NULL، وبما أن المؤشر temp لا يحقق هذا الشرط فسندخل في هذه الدوارة ؛ الأسطر من 7 إلى 10 لا تحوي أي شيء مهم ولكن في السطر 11 يتم إسناد العنصر الذي يشير إليه المؤشر first في القائمة المترابطة إلى المؤشر temp ويستمر تنفيذ هذه الدوارة حتى يصل المؤشر temp إلى المؤشر newlink(5) والذي لا يشير إلى لا شيء وبالتالي يخرج من الدوارة وينتهي تنفيذ هذه الدالة ، وبالطبع فإن أهم الأسطر في هذه الدالة هم الأسطر: 3 و 5 و11 .

التابع Find( ):

هذا هو تعريف هذه التابع:

```
1. void linklist::find(int f)
2. {
3. int m=1;link* temp=first;
4. while( temp != NULL ) {
5.  if (f==temp->number)
6. {
7. cout << "It is exisit\n";
8. cout <<"Number Of Course:\t" << temp->number << endl;
9. cout << "its degree:\t\t" << temp->degree << endl;
10.      cout << "its Hours:\t\t " << temp->hours << endl;
```

```
11.      cout  <<  "------------------------------------------
   \n";
12.      m++;
13.      break;}
14.      temp = temp->next;
15.      }
16.      if(m==1) cout << "\n\n(SORRY)....Not Exisit\n";
17.      }
```

من مهام هذه الدالة إيجاد رقم المادة المطلوب البحث عنها وبالتالي عرض بيانات هذه المادة أو ذلك المقرر الجامعي ، في السطر الأول تم الإعلان عن المتغير m **وتهيئته بالقيمة** 1 **، وأيضاً تم الإعلان عن مؤشر مؤقت من التركيب** link **وتهيئته بالمؤشر** first **، نفس طريقة التنفيذ في دالة** Display **، هي نفسها هنا إلا أن المميز في هذه الدالة هو وجود السطر** 5 **، حيث تقارن هذه الدالة بين رقم المادة الممرر وأرقام جميع المواد في القائمة المترابطة وفي حال وجدت الرقم المراد فإنها تدخل في تنفيذ جملة القرار** if **، وتطبع جميع بيانات تلك المادة ؛ والمميز هنا هو نهاية هذه الجملة حيث تقوم بزيادة المتغير** m **ينتهي تنفيذ الجملة** if **وينتهي معها التكرار** while **، أما في حال لم تجد هذه الدالة الرقم المراد فإنها لا تقوم بزيادة المتغير** m **وبالتالي فإن المتغير** m **يبقى على حاله ، مما يؤدي إلى نجاح جملة المقارنة في السطر** 16 **وتطبع الجملة الموجودة السابقة والتي لا تدل على وجود الرقم المراد إيجاده أو البحث عنه.**

الدالة- برنامج الإختبار- ( )main:

هذا هو برنامج الإختبار لهذه القائمة المرتبطة ، ولن نقوم الآن بشرحه فهو بسيط وسهل ولا يعتقد أنك لا تملك القدرة على فهمه.

```
1. int main()
2. {
3. linklist li;
4. int m;
5. int i=1;
6. char choice;
7. do
8. {
9. cout << "ENTER YOUR CHOICE:" << endl;
10.   cout << "(a)for entre data\t(b)for
   search\t(c)print\t(d)END:   ";
11.   cin >> choice;
12.   switch (choice)
13.   {
14.   case 'a':
15.   for (;;)
```

```
16.    {
17.    cout << "Enter the number of Course?:";
18.    cin >> m;
19.    if (m==0) break;

20.    li.additem(m);
21.    li.Enter();
22.    }
23.    break;
24.    case 'b':
25.    int n;
26.    for(;;)
27.    {
28.    cout << "Do you want to search?\t";
29.    cin >> n;

30.    if (n==0) break;
31.    cout << "\nJust wati a minute" << endl;
32.    li.find(n);
33.    }
34.    break;
35.    case 'c':
36.    li.display();break;

37.    case 'd':
38.    i=2;
39.    }
40.    }while (i==1);
41.    return 0;
42.    }
```

عيوب هذه القائمة :

بالرغم من الميزة لهذه القائمة وهي أنها أكثر مرونة من المصفوفات ، ونظراً لمرونتها الشديدة فلا حاجة لأن تكون هذه القائمة وفق ترتيب معين، لأن المميز بينها هو رقم المقرر الجامعي فقط ، وتستطيع زيادة تحميل المعامل [ ] ليصبح قادراً على التعامل مع هذه القائمة ، ولكن هناك بعض العيوب في هذه القائمة المترابطة ، فأولاً الدالة Display تقوم بعكس ترتيب هذه القائمة ، وأيضاً ففي الأساس ليس هناك ترتيب معين لهذه القائمة فأول مقرر جامعي تقوم بإدخاله يتم وضعه في الأخير ، ولربما كان من الأفضل ترتيب هذه المقررات حسب ترتيب واحد تصاعدياً أو تنازلياً حتى

نزيـد مـن سرعــة البرنـامج عــند التعــامل مـع أعــداد كبيـرة مـن المقـررات الجامعيــة وأيضـاً هــناك عيـب ثالـث فـي هـذه القائمــة ألا وأنهـا لا تسـتطيع الإشـارة إلـى العــناصر الأخـرى فهـي لا تشـير إلا إلـى عــنصر واحـد فقـط ، المثال القادم سيحل بعض من هذه المشاكل التي تراهـا أنـت صـغيرة ولكـن حينما تصل إلى التنفيذ فقد تكبر ولا تجـد لها حلاً أبداً.
سنقوم الآن بتطوير القائمـة المرتبطـة إلى شكل أفضل وسنترك لك أنت حـل المشاكل السابقـة والتي ربما قد تجد حلاً لها في المثال القادم.

قوالب الكائنـات:

بعـد أن تعرضـنا فـي وحـدة سـابقة لقوالـب التوابـع ، نجـد هــنا أنــه بإمكاننـا استخدام القوالب في الكائنـات.
لقد قمت بتأجيل الحديث عـن هذا الموضوع حتى نفهـم الفائدة من القوالـب ، وهـنا أحد الفوائد المهـمـة للقوالب.
لنفرض أنك سـتقوم بتطوير stack شبيه بالذاكرة الموجـودة فـي الحاسـب ، ، تذكر أن هذه الذاكرة ليست شبيها بالمصفوفات من أي ناحيـة لذلك لن تخـزن عـناصرها بشكل مرتب أو متسلسل. انظر إلى هذا المثال الكـودي الشـهير:

```
1. #include <iostream>
2. using namespace std;
3.
4. const int max=20;
5. class Stack
6. {
7. int st[max];
8. int top;
9. public:
10.   Stack(){top=-1;}
11.   void push( int element ){st[++top]=element;}
12.   int pop() { return st[top--];}
13.   };
14.
15.   int main()
16.   {
17.         Stack temp;
18.
19.      temp.push(10);
20.      temp.push(11);
21.      temp.push(12);
22.      temp.push(13);
23.      temp.push(14);
24.      cout << "First:\t " << temp.pop() << endl;
25.      cout << "Second:\t " << temp.pop() << endl;
```

```
26.       cout << "third:\t " << temp.pop() << endl;
27.          cout <<"fourth:\t" << temp.pop() << endl;
28.          cout << "fifth:\t" << temp.pop() << endl;
29.
30.          return 0;
31.    }
32.
```

ليس هـناك من كثير لشرحـه ، لذلك سأترك لك مهـمـة فهـم هذا المثـال لأنـه سيفيدك في وحدة قادمـة (مكتبة القوالب القياسية).
المهـم في هذا الأمر أن الصنف stack لا يقبل بيانات إلا من النوع int ولا يقبل غيرها ، عـن طريق القوالب سنقـوم بجعل هذا الصنف يقبـل أي شـيء حتـى لو كانت أصنافاً أخرى قام مستخدم آخر بكتابتها.

انظر الآن إلى الصنف Stack بعـد استخدام القوالب معه.

CODE

```
1. const int max=20;
2. template <class T> class Stack
3. {
4. T st[max];
5. int top;
6. public:
7. Stack(){top=-1;}
8. void push( T element ){st[++top]=element;};
9. T pop() ;
10.   };
```

بإمكان الآن الصنف Stack التعامل مع جميع أنـواع البيانات، بواسطـة النـوع T ، الذي سيتم التحقق منـه من خلال إنشاء كائن من هذا الصنف.
انظر إلى السطر 9 لقد تركـنا تعريف هذا التابع حتى تفهـم كيـف يـتم تعريـف توابع قالب صنف ما ، انظر إلى تعريف هذا التابع:

```
1. template <class T> T Stack<T>::pop()
2. {
3. return st[top--];
4. }
```

انظر إلى السطر الأول ورأس هذا التابع ، تجد أنـه أولاً تـم ذكـر القالـب وهـو template <class> ، ثم تـم ذكـر نـوع القيمـة المعادة وهـي T ، ثـم اسـم الصنف الذي ينتمي إليه القالب وهـو Stack ، ثـم بـين قوسـين حـادين اسـم البيانات وهـي <T> ، ثم أربع نقاط ثم اسـم التـابع. بهـذه الطريقـة بإمكانـك تعريف قوالب أي توابع صنف آخر.

بقي الآن أن أذكر كيفية استخدام هذا الصنف ، انظر إلى التابع main() :

```
1. int main()
2. {
3.     Stack <int> temp;
4.
5.     temp.push(10);
6.     temp.push(11);
7.     temp.push(12);
8.
9.     cout << "First:\t " << temp.pop() << endl;
10.    cout << "Second:\t " << temp.pop() << endl;
11.    cout << "third:\t " << temp.pop() << endl;
12.
13.    return 0;
14.    }
```

انظر إلى طريقة استخدام هذا القالب والتي تختلف عن استخدام قوالب التوابع ، في السطر 3 . حيث يجب علينا ذكر نوع البيانات التي نود تخزينها بين قوسين حادين < > ،الآن سيكون بمقدورنا استخدام هذا الصنف مع صنف نقوم بإنشاءه نحن ، ربما ستود استخدام صنف الأعداد الكسرية Fraction الذي كتبناه في وحدة سابقة من الكتاب.
بإمكاننا كتابة أيضاً قالب للتركيب struct وطريقة إستخدامه هي نفس طريقة استخدام الصنف class .

إستخدام القوالب مع القائمة المرتبطة:

الآن سنأتي لإحدى فوائد القوالب وهي استخدامها في تخزين البيانات أي في القوائم المرتبطة.
ارجع إلى مثال القائمة المرتبطة في هذه الوحدة ، وانظر أين سنقوم استخدام هذه القوالب.
سنقوم الآن بتطوير مفهوم القوائم المرتبطة وسنحاول كتابة قائمة بدائية باستخدام القوالب ، هذه الوحدة تعتبر مقدمة لك حتى تطور إمكاناتك في القوائم المرتبطة وبنى المعطيات ولن تقوم هذه الوحدة بمعالجة مواضيع متقدمة نسبياً.
سنقوم بتطوير القائمة السابقة (المقررات الجامعية) ولكن قبلاً فعلينا شرح استخدام القوالب مع القوائم بشكل أفضل.
القائمة التي سنكتبها عبارة عن قائمة تخزن نوعاً واحداً من البيانات هو عدد أو حرف وسترتبط عناصر هذه القائمة ببعضها البعض.
انظر إلى هذا المثال الكودي الطويل:

CODE

```
1. #include <iostream>
2. using namespace std;
3. /**********************************/
```

```
4. template <class T>struct link
5.    {
6.    T element;
7.    link* next;
8.    };
9. /*********************************/
10.   template <class T> class linklist
11.      {
12.      private:
13.         link <T>* first;
14.      public:
15.         linklist()
16.         { first = NULL; }
17.         void additem(T d);
18.         void display();
19.      };
20.   /*********************************/
21.   template <class T> void linklist<T>::additem(T x)
22.   {
23.   link<T> *newlink = new link<T>;
24.   newlink->element = x;
25.   newlink->next = first;
26.   first = newlink;
27.   }
28.   /************************************/
29.
30.   template <class T> void linklist<T>::display()
31.      {
32.      link<T> *temp = first;
33.      cout << "\n\n----------------------------------------\n";
34.      while( temp != NULL )
35.         {
36.            cout << endl <<" number:\t" << temp->element;
37.         temp = temp->next;
38.         }
39.      }
40.   /******************************************/
41.
```

```
42.  int main()
43.  {
44.  linklist<double> li;
45.  int m;
46.  int i=1;
47.  char choice;
48.  do
49.  {
50.  cout << "ENTER YOUR CHOICE:" << endl;
51.  cout << "(a)for entre data\t(b)for print\t(c)END:   ";
52.  cin >> choice;
53.  switch (choice)
54.  {
55.  case 'a':
56.  for (;;)
57.  {
58.  cout << "\nEnter the element you want to add it:";
59.  cin >> m;
60.  if (m==0) break;
61.
62.  li.additem(m);
63.  }
64.  break;
65.  case 'b':
66.  li.display();break;
67.
68.  case 'c':
69.  i=2;
70.  }
71.  }while (i==1);
72.  return 0;
73.  }
```

- في السطر الرابع قمنا بجعل التركيب link قالباً وفي السطر السادس يستطيع هذا المتغير تخزين أي نوع من البيانات.
- في السطر السابع قمنا بوضع المؤشر الذي سيقوم بربط هذه القائمة.
- في السطر العاشر قمنا بجعل الصنف linkedlist قالباً وبالتالي فسيستقل جميع الأنماط في انظر إلى السطر 13 وكيفية الإعلان عن هذا المتغير.

• لا جـديد في بقية الكـود.

استخدام القوالب مع قائمـة أكثر تعقيداً:

سنتعرض في هـذه الفقـرة علـى كيفيـة ربـط قائمـة مرتبطـة ببعضـها؛ تكـون عـناصرها أصـنافاً قمـت أنـت بكتابتهـا ، وسـيلتنا إلـى هـذا هـي كتابـة صـنف مستقل تمام الاستقلال عـن أي تعديل من الكائنـات الاخـرى ، سـنقوم أيضـاً بزيـادة تحــميل معامـل << ومعامـل >> ، أمـا عـــن التركيـب linke والقائمـة المرتبطة linkedlist فلن نتعرض لهـما بـأي شـيء ولكـن ربمـا تـود أنـت فـي المستقبل إضافة بعض الخدمات إليها مثل البحث وحذف عــنصر ما والترتيـب والتعـديل وغير ذلك ولكن جميع هذه الإضـافات لـن تكـون بسـبب اسـتخدامك لأصناف جـديدة بل لزيادة النواحي الإجرائية للقائمة التي تكتبها.

أيضاً لن يكون هذا المثال مشابهاً تمام الشبـه بالكـود الأول أو المثال الأول بل تركنا لك بعض المشاكل لكي تقوم أنت بحلها وحلها بسـيط للغايـة ولا يحتـاج منك سوى قليل من التفكير ، من هـذه المشـكلة مشـكلة إظهـار رقـم المـادة والتعـامـل مـع هـذه المـواد علـى أسـاس رقـم المـادة ، أيضـاً بإمكانـك زيـادة النواحي الإجرائيـة للقائمـة التي كتبناها لتصبح قائمة يعتمد عليها.

سنقوم الآن بكتابة صنف الطالـب الـذي بإمكانـك تخـزينــه فــي القائمــة فــي المثال السابق.

CODE

```
1. class courses
2. {
3. int number;
4. float grade;
5. int hours;
6.
7.  public:
8.    courses();
9.    courses(int a);
10.         courses (const courses& rhs);
11.         int getNumber()const;
12.         float getGrade()const ;
13.         int getHours()const ;
14.         setNumber(const int a){number=a;}
15.         setHours(const int a) {hours=a;}
16.         setGrade(const float a){grade=a;}
17.         courses &operator= (const courses &);
18.         friend ostream &operator << (ostream& ,const courses &);
19.         friend istream &operator >> (istream& E, courses &temp);
20.
21.   };
22.   courses::courses (): number(0),grade(0),hours(0){}
```

```
23.   courses::courses (int a):number(a),grade(0),hours(0){}
24.   courses::courses (const courses& rhs)
25.         {number=rhs.getNumber();
26.         grade=rhs.getGrade();
27.         hours=rhs.getHours();
28.         }
29.
30.   int courses::getNumber()const {return number;}
31.   float courses::getGrade()const {return grade;}
32.   int courses::getHours()const {return hours;}
33.
34.   courses& courses::operator= (const courses& rhs)
35.         {
36.         if(this==&rhs) return *this;
37.         number=rhs.getNumber();
38.         grade=rhs.getGrade();
39.         hours=rhs.getHours();
40.         return *this;
41.         }
42.
43.   ///////////////////////////////////////
44.          istream& operator >> ( istream& E, courses& temp)
45.         {     float i=0;int j=0;
46.               cout << "Enter its grade:";
47.            cin >> i;temp.setGrade(i);
48.               cout << "Enter its hours: ";
49.           cin >> j;temp.setHours(j);
50.         return E;
51.         }
52.   ////////////////////////////////////////
53.         ostream &operator << (ostream& D , courses &temp)
54.         {
55.               D <<"Number Of Course:\t" << temp.getNumber() <<
   endl;
56.               D << "its degree:\t\t" << temp.getGrade() << endl;
57.               D << "its Hours:\t\t " << temp.getHours() << endl;
58.         D << "---------------------------------------------\n";
59.
```

```
60.          return D ;
61.          }
```

انظر يوجد هـناك ثلاث متغيرات خاصـة وهـي التي تحدد الحالة الداخلية لهـذا الصنف والواجهـة يبلغ عـدد أعضائها 10 توابع بالإضافة إلى تابعين صديقين.
حتى يستطيع هذا الصنف العـمل في أي قائمـة سواءً هذه القائمـة أو غيرها فيجب أن يكون تابع بناء النسخـة معرفاً حيـث أن الإعـلان عــنه موجـود فـي السطر 10 وتعريفـه موجود في السطر 24.
أيضاً لا بـد أن يكـون معـاملي << و >> معرفـان ضـمن هـذا الكـائن حيـث أن الإعلان عـنهـما موجود في السـطر 18 و 19 كتابعـان صـديقان أما تعريفهـمـا ففي السطران 44 و 53 .
لاحظ الكائن courses وقارن بين مدى الشبـه بينه وبين المثال الأول في هذه الوحدة.
الآن بقي عليك تطوير القائمـة حتى تصبح شجرة معقـدة ، وفي حال سـئمت من صنع القوائم فاذهب إلى وحدة مكتبة القوالب القياسية فسوف تجـد فيـه مكتبات مختصـة فقط بالقوائم المرتبطـة.
الآن انظر إلى التابع main() وكيفية اختبار هذه القائمـة وهذا الصنف.

CODE

```
1. int main()
2. {
3.    linklist<courses> course;
4.    courses a;
5.    char b;
6.
7.    do
8.       {
9.       cin >> a;
10.          course.additem(a);
11.          cout << "\nAdd another (y/n)? ";
12.          cin >> b;
13.          } while(b != 'n');
14.       course.display();
15.       cout << endl;
16.       return 0;
17.       }
```

لا جـديد في التابع main( ) ، ولكن هـناك أمر مهـم أود الإشارة إليـه وهــو أن هذا المثال الأخير لن يعـمل إذا استخدمت المكتبة iostream الجديـدة بسـبب وجود مشاكل في المعامل >> ، وحتى يعـمل فعليـك التخلـي عــن مسـاحة الأسماء std والتعامل مع المكتبة القديمة iostream.h .

التعامل مع الاستثناءات

Handling Exceptions In C++

بدايـة:

لأي مبرمج يود أن يأخذ البرمجـة على محمل الجد أن يركز دائماً على الأخطاء التي من الممكن أن تحدث في برنامجـه ، هذه الأخطـاء والمشـاكل لهـا صـور مختلفـة للغاية.

أول هـذه الأخطـاء وأخطرهـا: هـي المنطـق الضـعيف للبرنـامج ، سـيؤدي البرنـامج مـا يـود المسـتخدم القيـام بـه ، إلا أن هـناك خطـأ ربمـا إحـدى الخوارزميات التي يستخدمها البرنامج ، إما أنها غيـر قـادرة علـى التعامـل مـع بعض الحالات الاستثنائية .

أيضاً هـناك أخطاء أخرى منها وقد تتبع النموذج السـابق عـدم قـدرة البرنـامج على التعامل مع حالات غير متوقعـة ، فماذا لـو قـام المسـتخدم بإدخـال رقـم هاتفـه المنزلي مكان متغير حرفي أو سلسلة حرفيـة.

ومـن الأخطـاء أيضـاً الأخطـاء الخاصـة بالمؤشـرات أو الـذاكرة وأيضـاً لـو عـند التعامـل مـع الملفـات ، مـاذا لـو كـان الملـف الـذي يبحـث عـنه البرنـامج غيـر موجود.

ما هـو الاستثناء:

حينما يتسلم البرنامج مهـمـة ما ، مثلاً القيـام بعـملية الجمـع أو الطـرح أو أي شـيء برمجـي آخـر ويفشـل فـي أداء مهـمته ، فإنــه يقـوم بإرسـال هـذه المهـمـة أو هذا الخطأ إلى نظام التشغيل ، والذي سيحاول معرفـة ماذا يفعل في بعض الحالات يعرف (مثل حالة القسمة على صفر) وفي بعض الحـالات يقـوم بإيقـاف البرنـامج عــن الحـالات وفـي بعـض الحـالات ينطلـق البرنـامج انطلاقة الصاروخ في تكرارات لانهائيـة قـد تنتهـي إن لـم تسـيطر عليها علـى إنهاء جلسة عـملك على نظام التشغيل.

كمبـرمج لا بـد أن تعـرف مـا هــي الأخطـاء التـي سـيقوم المتـرجم بإرسـالها وبالتالي منعــه مـن إرسـالها إلـى نظـام التشـغيل والسـيطرة عليهـا، وتكـون السيطرة عليها بأحد الخيارا ت التالية:

- إيقاف البرنامج.
- إصدار تنبيه بوجود المشكلة والخروج من البرنامج بشكل آمن.
- تنبيه المستخدم والسماح له بمعالجة المشكلة ومواصلة العمل.
- معالجة المشكلة بدون تدخل المستخدم ومواصلة العمل.

التعامل مع الاستثناءات :

تزودك لغـة السـي بلـس بلـس بـثلاث كلمـات مفتاحيـة ، سـنتعرف عليهـا حـالاً ولكن حاول الربط بين مضمون الفقرة السابقة وهذه الفقرة:

1- توقع الاستثناءات try :

الكتلـة try تخبـر المتـرجم أو البرنـامج بأنـك تتوقـع أن يكـون هــنا أحـد الاسثناءات أو الأخطاء ، والصيغـة العامـة لهذه الكتلة هـي كالتالي:

```
try {
statement1;
statement2;
.....etc;
}
```

2-كتلة معالجة الأخطاء:
حينما تقوم الكتلة try بإخبار البرنامج أو المترجم عن الأخطاء المتوقعة في هذا القسم من الكود وفي حال بالفعل حدث استثناء أو خطأ فإنها تنتقل إلى كتلة معالجة الأخطاء وهي الكتلة catch ، يوجد أنواع مختلفة من كل catch ، حيث أن لكل خطأ كتلة catch خاصة به، والصيغة العامة لهذه الكتلة هي كالتالي:

```
catch (Argument)
{
statement1;
statement2;
//What to do about this exception
}
```

3- إلقاء الاستثناءات:
هناك كلمة ثالثة وهي مجرد كلمة مفتاحية ليست عبارة عن كتلة ، هذه الكلمة تعمل ضمن الكتلة try (وفي بعض الحالات ضمن الكتلة catch) وتقوم بإلقاء استثناء أي تطلب من البرنامج الخروج من الكتلة try والانتقال إلى الكتلة catch .

مثال عملي:
سنقوم الآن بكتابة برنامج نعالج فيه الاستثناءات المتوقع حدوثها، سنقوم بكتابة كود يقوم المستخدم فيه بإدخال السنة ، وعلينا الآن أن نتوقع الاستثناءات أو الأخطاء المتوقع حدوثها وبكل حال فلن يكون هناك إلا نوعين من الأخطاء:

1- أن يدخل المستخدم سلسلة حرفية.
2- أن يدخل المستخدم عدداً أقل من الصفر أو مساوياً له ، لأنه لا وجود لأي سنة سالبة أو تساوي الصفر.

CODE

```
1- #include <iostream>
2- using namespace std;
3-
4- int main()
5- {
6-     int TheYear;
7-
8-     cout << "TheYear ";
9-     cin >> TheYear;
10-
```

```
11-         try {
12-                 if(TheYear <= 0)
13-                         throw;
14-                 cout << "\nTheYear: " << TheYear << "\n\n";
15-         }
16-
17-         catch(...){
18-                         }
19-
20-         cout << "\n";
21-
22-         return 0;
23-     }
```

بداية قمنا بالتصريح عن المتغير TheYear ، والذي سنحاول من خلال هذا الكود حل المشكلتين الخاصة به.
في السطر 11 يدخل البرنامج في الكتلة try والتي نتوقع فيها أن يكون هناك أخطاء وتنتهي هذه الكتلة في السطر 15 .
المستخدم يقوم بإدخال رقم السنة قبل الكتلة try وبالتالي فحينما لا ينجح هذا الإدخال سينتقل التنفيذ إلى الكتلة catch في السطر 17 وسيقوم نظام التشغيل بعرض رسالة عليك تخبرك بوجود أخطاء وسيقوم بإيقاف البرنامج التلقائي.
لو نجح الإدخال ولكن كان أصغر من الصفر أو مساوياً له فإن الجملة التي تحاول حل هذه المشكلة موجودة في السطر 12 حيث تقوم الكلمة throw في حال نجاح الجملة if بإلقاء استثناء.
هل ترى الجملة catch ، وخاصة النقاط الثلاث بين القوسين (...) ، هذه الجملة تعني جميع الاستثناءات أي أن الكتلة catch (الموجودة حالياً) هي كتلة اصطياد استثناءات عامة هناك المزيد من الكتل أيضاً قد تضعها ولكن من المفضل أن تضعها قبل هذه الكتلة العامة لأنك إن وضعتها بعد الكتلة العامة فلن تستطيع تلك الكتل اصطياد الإستثناءات الخاصة بها لأن جميع الاستثناءات ستصطادها الكتلة العامة.

في الحقيقة فإن الكود السابق لا يتعامل مع الاستثناءات بالشكل الذي نريده فوجود الكلمة throw ، دون أي عبارات أخرى تعني استدعاء المدمر ، أي أن نظام التشغيل سيقوم سيقوم بعرض رسالة ويقوم بتدمير البرنامج، ما رأيك الآن لو نقوم بتطوير الكود السابق ليصبح قادراً على التعامل مع الاستثناءات بشكل مقبول :

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
```

```
5. {
6.    int TheYear;
7.
8.    cout << "TheYear ";
9.    cin.exceptions(cin.failbit);
10.
11.
12.         try {
13.               cin >> TheYear;
14.               if(TheYear <= 0)
15.                     throw "Hellow" ;
16.               cout << "\nTheYear: " << TheYear << "\n\n";
17.         }
18.
19.
20.         catch(...){
21.               cerr << "\nSomthing is not right\n";
22.         }
23.
24.         cout << "\n";
25.
26.         return 0;
27.   }
```

المهم في هذا الكود المعدل حالياً هو السطر 9 ، حيث يقوم هذا السطر بإعداد الكائن cin لإلقاء الأخطاء والاستثناءات في حال وقوعها وبالتالي التعامل معها ، إذا لم تقوم بكتابة ذلك السطر فلو قام المستخدم بإدخال سلسلة حرفية مكان متغير فسيقوم نظام التشغيل بتدمير البرنامج وعرض رسالة بذلك .

السطر 15 قام بوضع سلسلة حرفية بعد الكلمة المفتاحية throw ، هذه السلسلة الحرفية تعد بمثابة الخطأ الذي سترميه الكلمة throw لتتلقاه إحدى كتل catch ، وبما أنه لن توجد أي كتلة catch قادرة على التعامل مع هذه السلسلة الملقاة فستمضي إلى الكتلة catch العامة ، في السطر 20 يدخل البرنامج إلى الكتلة catch في حال وقوع أي استثناء .

انظر إلى السطر 21 هذا السطر يقوم بعرض سلسلة حرفية ، وهو لا يقوم بعرضها بواسطة الكائن cout القياسي بل بكائن cerr وهذا الكائن خاص بإخراج رسائل الأخطاء ، تستطيع الاستعاضة عنه بكتابة الكائن cout ، ولكن التركيب الداخلي لهذا الكائن يمكنه من عرض رسائل الأخطاء بشكل أفضل من cout حيث لن يكون هناك مجهود أقل للحاسب في هذه الناحية.

كتل catch متعددة:

قد ترغب في بعض الأحيان بأن يكون هناك تعامل مختلف لبعض الاستثناءات، توفر لك لغة السي بلس بلس فعل هذا الشيء ، سنقوم الآن بإعادة كتابة الكود الأخير وسيكون هناك تعامل مختلف حينما يدخل المستخدم الرقم 0 أو قيمة أقل منه ، وبذلك يكون برنامجنا أفضل:

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.    int TheYear;
7.
8.    cout << "TheYear ";
9.    cin.exceptions(cin.failbit);
10.
11.
12.         try {
13.               cin >> TheYear;
14.               if(TheYear <= 0)
15.                     throw "Bad Value" ;
16.               cout << "\nTheYear: " << TheYear << "\n\n";
17.         }
18.
19.         catch (char *Error){
20.               cerr << Error << endl;
21.         }
22.
23.
24.         catch(...){
25.               cerr << "\nSomthing is not right\n";
26.         }
27.
28.         cout << "\n";
29.
30.         return 0;
31.   }
```

لقد قلنا أننا سنقوم بتطوير الكود السابق ليتعامل بشكل مختلف في حال أدخل المستخدم الرقم 0 **، الآن انظر إلى الجملة** if **في السطر** 14 **والأمر الذي ستقوم بتطبيقه في حال نجاح الاختبار وهو الموجود في السطر** 15.

في السطر 15 **تقوم الكلمة** throw **بإلقاء سلسلة حرفية وهي** Bad Value **حينما يقوم المستخدم بإدخال الرقم** 0 **أو قيمة أقل منه ستقوم الكلمة** throw **بإلقاء هذا الاستثناء.**

يقوم المترجم بالبحث ضمن كتل catch **والأمر أشبه بما يمكن القول أنه استدعاء توابع ذات زيادة تحميل ، وسيجد المترجم الكتلة المناسبة في السطر** 19 .

كما قلنا ألقت الكلمة throw **سلسلة حرفية والكتلة** catch **الموجودة في السطر** 19 **تستقبل السلاسل الحرفية الملقاة عبر الكلمة** throw **، انظر إلى التعبير:**

```
catch (char *Error){
```

حيث أن الكتلة catch **تستقبل المؤشرات الحرفية وبالتالي فإن التنفيذ سينتقل إليها.**

يقوم السطر 25 **بعرض السلسلة الحرفية الملقاة ثم ينتهي تنفيذ البرنامج.**

الخلاصة:
تقوم الكلمة throw **بتوليد الاستثناءات، في حال توافق نمط البيانات التي تلقيه الكلمة** throw **مع أي من كتل** catch **فإن التنفيذ سينتقل إلى هذه الكتلة.**

مثال عملي:

بالنظر إلى الوحدات والمواضيع التي اجتزتها فاعتقد أن بإمكانك فهم هذا المثال والذي سيتعامل مع جميع أنواع المشاكل المتوقعة والغير متوقعة ، يقوم هذا الكود بجعل الطالب يقوم بإدخال عدد مواده ثم يقوم بإنشاء مصفوفة ديناميكية لهذه المواد ويطلب من المستخدم إدخال جميع درجات مواده ويحسب مجموع درجات هذه المواد والمتوسط الحسابي لها ، ولا يكتفي فقط بذلك بل في حال حدوث أي خطأ في البرنامج سواء عند إدخال المواد فسيقوم البرنامج بإعلام الطالب بهذه المشكلة ويطلب منه إعادة الإدخال ، تصور معي مقدار التحدي الذي سيواجهه هذا الكود (الكود بشكل عام بسيط للغاية ولا يعد كوداً مقارنة بالأكواد الاخرى العملاقة ولكني حاولت إثارة القارئ الكريم حول التحدي البسيط الذي سيواجهه في هذا الكود):

CODE

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.    cin.exceptions(cin.failbit);
7.    float *Course;
8.    int NumberOfCourse;
9.    float Total;
```

```
10.         float Avg;
11.
12.
13.         try {
14.               cout << "please Enter the number of course:\t";
15.               cin >> NumberOfCourse;
16.               Course=new float[NumberOfCourse];
17.               for (int i=0;i<NumberOfCourse;i++)
18.               {
19.                     try{
20.                     cout << "Enter the grade of couse" << i+1
21.                     << ": \t";
22.                     cin >> Course[i];
23.
24.                     Total=+Course[i];
25.                     }
26.                     catch(...)
27.                     {
28.                           cerr << "\nBad Value\n";
29.                           cin.clear();
30.                           char Bad[5];
31.                           cin >> Bad;
32.                           i--;
33.
34.                     }
35.
36.               }
37.               Avg=Total/NumberOfCourse;
38.               cout << endl;
39.               cout << "The Total of grade:\t" << Total;
40.               cout << "\nThe Avg:\t" << Avg;
41.               cout << endl;
42.         }
43.         catch(...)
44.         {
45.               cerr << "Somthing not right" << endl;
46.   }
47.         return 0;
```

```
48.    }
```

عليك الانتباه جيداً لما يحتويه هذا الكود فلقد قمت بتضمين بعض المفاهيم الجديدة فيه سنتعرض لأهمها حالاً:

- في السطر 6 يقوم بالطلب من الكائن cin ، الاستعداد لإلقاء الاستثناءات في حال وقوع أي أخطاء.
- في الأسطر 7-10 تم الإعلان عن أربع متغيرات هي المتغيرات التي سيحتاجها البرنامج والمتغير الموجود في السطر 7 هو عبارة عن مؤشر سيقوم بإنشاء مصفوفة ديناميكية الحجم حتى يقوم المستخدم فيها بإدخال حجم المواد التي يريد حسابها.
- في السطر 13 يدخل البرنامج إلى الكتلة try . تذكر أنها أول كتلة try .
- في السطر 16 يقوم البرنامج بإنشاء مصفوفة المواد ، في حال وقوع أي استثناء في الأسطر من 13 إلى 16 فسينتقل التنفيذ إلى السطر 26 .
- يدخل البرنامج في الحلقة for في السطر 17 والتي سيقوم المستخدم من خلالها بإدخال المواد ، الآن علينا أن نقوم بإنشاء كتلة معالجة الأخطاء والسبب في ذلك هو أنه لو قام المستخدم بإدخال بيانات خاطئة حينها سيقوم البرنامج بحل تلك المشكلة وسيطلب من الطالب الاستمرار في إدخال البيانات.
- في السطر 19 تم الدخول في كتلة try جديدة هذه الكتلة لن تعمل إلا في الحلقة for . أي الأخطاء والاستثناءات التي ستقع ضمن الحلقة for .
- في الأسطر 20-24 يقوم البرنامج بعلاج إدخالات المستخدم ويحسب من خلالها مجموع درجاته ، الآن ماذا لو قرر المستخدم إدخال اسمه هو في السطر 22 ، الآن مالذي سيحدث ، الذي سيحدث هو أن هذه الأحرف ستبقى عالقة فمثلاً لو أدخل (abs) فإن هذه الاحرف ستبقى عالقة في البرنامج وسيكون عليك الآن محاولة أخذ هذه الأحرف من المتغير العددي ومحاولة إسنادها في متغير حرفي وبالتالي التخلص منها بشكل آمن ، الآن سينتقل التنفيذ إلى كتلة catch التي ستعالج إدخالات المستخدم الخاطئة.
- يقوم السطر 29 باستدعاء التابع العضو clear حتى يقوم بتنظيف الإدخالات الخاطئة.
- في السطر 30 تم الإعلان عن مصفوفة حرفية.
- في السطر 31 يتم أخذ الأحرف العالقة التي أدخلها المستخدم خطأً ويتم وضعها في السلسلة الحرفية التي تم الإعلان عنها في السطر 30.
- في السطر 32 تتم إنقاص قيمة دليل الإدخال أو عداد الحلقة for قيمة واحدة وبالتالي فلن تنتقل الحلقة for إلى الإدخال الجديد بل ستبقى في الإدخال الذي أدخله المستخدم خاطئاً حتى يعيد إدخاله من جديد.
- بقية الأسطر واضحة ولا غبار عليها ولا جديد فيها.

الكائنات والاستثناءات:

هناك مشكلة كبيرة في حال استخدمنا الاستثناءات حسب الطريقة السابقة وهي أنه لن يمكننا التعامل مع جميع حالات الاستثناء كما رأينا من أكواد

سابقة أننا ألقينا سلسلة حرفية بواسطة throw حتى تصطادها جملة catch تتعامل مع السلاسل الحرفية ، لاحظ أننا لو كتبنا throw أخرى وألقينا سلسلة حرفية فإن نفس الكتلة السابقة catch ستتعامل معها قد تقول سنستخدم int و char و float ... وغيرها ولكن كل ذلك سينتهي عما قريب وقد تحصل على ربما 10 كتل catch وهذه الكتل لن تستطيع التعامل مع أنواع أخرى ملقاة ، جميع البرامج التجارية يجب أن تكون قادرة على جميع حالات الأخطاء هذه ، يكمن الحل الوحيد هو بإستخدام الكائنات وتخصيصها ككائنات استثناء وفي حال وقع أي استثناء فإننا نلقي بذلك الكائن أي سنتعامل مع تلك الأصناف أو الكائنات على أنها int و float ... إلخ مع اختلاف أننا سنقوم بإنشاء أي عدد من تلك الأصناف وبالتالي التعامل مع الأخطاء بشكل يسير وميسر ، بل إن هذا الأسلوب يعتبر أسلوباً أفضل من الأسلوب السابق حتى لو كان عدد الاسثناءات المتوقعة في برنامجنا قليلة والسبب في ذلك يعود إلى مرونة هذا الأسلوب وإلى أن الكائن سيصبح أكثر استقلالية لأنه يحوي استثناءات عند وقوع الاخطاء.

سنقوم الآن بكتابة صنف Stack يحوي مصفوفة تحوي الأعداد من 0 إلى 100 ، وسيكون هذا الصنف مستقل بحث يستطيع التعامل مع الإدخالات الخاطئة من قبل المستخدم.

CODE

```
1. #include <iostream>
2. using namespace std;
3. const int element=100;
4. class Stack
5. {
6.    int array[element];
7. public:
8.    Stack(){ for(int i=0;i<element;i++)
9.          array[i]=i;}
10.         getElement(int x)
11.         {
12.               if (x>100)
13.                     throw xThrow();
14.               else if (x<0) throw sThrow();
15.               else return array[x];
16.         }
17.         class xThrow{};
18.         class sThrow{};
19.   };
20.
21.   int main()
22.   {
23.         int i=0; Stack a;
```

```
24.         for(;;){
25.         try{
26.         cout << "Choise The element\n";
27.         cin >> i;
28.         cout << "The element:\t" << a.getElement(i) << endl;
29.         }
30.         catch(Stack::sThrow){
31.               cout << "Small Element\n";
32.         }
33.         catch(Stack::xThrow){
34.               cout << "Big Element\n";
35.         }
36.         }
37.
38.
39.         return 0;
40.   }
```

- انظر إلى الصنف Stack ، هذا الصنف يتحكم في مصفوفة تحوي 100 عنصر، ويمنح مستخدم الصنف القدرة على محتويات أي عنصر في هذه المصفوفة وفي حال كان الإدخال خاطئاً أو طلب المستخدم رقماً أكبر أو أصغر فإنه يستطيع التعامل معه.
- ابتدأ تعريف الصنف Stack من السطر 4 إلى السطر 19.
- انظر إلى الأصناف التي عرفت ضمن الصنف Stack ، وهما xThrow و sThrow في السطرين 17 و 18 .
- التابع العضو getElement( ) ، يستقبل عدد صحيح كبارامتر له ويحاول إيجاد العنصر المناسب ، يتعامل السطر 12 مع الأرقام التي أعلى من 100 حيث لا وجود لهذه العناصر في المصفوفة ، ويقوم بإلقاء استثناء وهو xThrow ، يتعامل السطر 14 مع الأرقام التي أقل من 100 حيث لا وجود لهذه العناصر في المصفوفة ويقوم بإلقاء استثناء وهو sThrow ، في السطر 15 يتعامل الصنف مع الأعداد الصحيحة ويقوم بإعادة العنصر المطلوب.
- الآن دعنا نلقي نظرة على التابع main( ) ، وقد تم تعريف المتغير i من النوع int وأيضاً تم تعريف الكائن a من الصنف Stck. وكل ذلك في السطر 23.
- يدخل البرنامج في حلقة for أبدية والسبب في ذلك هو سبب تعليمي حتى تستطيع أنت إدخال أي أعداد تريدها وسيقوم البرنامج بمعالجتها ، وذلك في السطر 24. وتنتهي هذه الحلقة في السطر 36 .
- يدخل البرنامج فوراً في كتلة try في السطر 25 ، ثم في السطر 27 يطلب البرنامج من المستخدم إدخال رقم العنصر الذي يود استدعاءه .

- يقوم السطر 28 باستدعاء التابع العضو getElement وتمرير العدد الذي أدخله المستخدم حتى يستطيع التابع استدعاء العنصر المناسب.
- إذا كان العدد الممرر أكبر من 100 فسيتم إلقاء استثناء في السطر13 وينتقل التنفيذ إلى كتلة catch الموجودة في السطر 33 والتي تطبع جملة تخبر المستخدم بأنه أدخل عنصر كبير ، لاحظ كيف تقوم عبارة catch باصطياد الأخطاء حيث أن الأخطاء التي تصيدها هي من النوع Stack::xThrow .
- يحدث نفس الشيء بالنسبة للإدخالات أصغر من 100 ويتعامل البرنامج معها بإلقاء استثناء من النوع Stack::sThrow والتي تصطاده الكتلة catch الموجودة في السطر 33 .
- إذا أدخل المستخدم عدداً صحيحاً فلن يحدث أية مشاكل وسيتم إعادة العنصر المناسب.
- سواءً كانت الإدخالات صحيحة أو غير صحيحة فسيعود البرنامج للطلب منك لإدخال عدد آخر ولن يتوقف أبداً حتى تقوم أنت بإغلاق البرنامج بنفسك ، قد تود أيضاً إضافة إستثناء يمكنك من الخروج من البرنامج.
- هناك أيضاً بعض الملاحظات في هذا المثال ألا وهي أنه لن يتم استدعاء تابع الهدم أبداً في هذا البرنامج حتى مع إلقاء استثناء.
- هناك أيضاً ملاحظة جديرة بالذكر ، وهي أن الصنف Stack يقوم بإلقاء استثناءات هي في الأساس أصناف وليست كائنات ، تذكر هذه النقطة جيداً ، وهذه الأصناف تم تعريفها ضمن تعريف الصنف Stack .

الاستفادة من كائنات الاستثناءات:

بامكانك أيضاً إنشاء كائن والاستفادة من خدمات هذا الكائن الذي ستلقيه ، وأيضاً تذكر لن تقوم بإلقاء استثناء كائن بل ستقوم بإلقاء استثناء صنف ، لن نتعمق كثيراً في هذا الموضوع بل كل ما سنقوم به هو إعطاؤك مقدمة واسعة للاستثناءات ، تذكر بإمكانك استخدام جميع مميزات الكائنات من وراثة وتعدد أوجه وتوابع افتراضية أو ظاهرية وغير ذلك. سنقوم بتطوير المثال السابق حتى يصبح قادراً على التعامل مع الاستثناءات بواسطة نفسه دون التدخل من كتلة catch .

CODE

```
1. #include <iostream>
2. using namespace std;
3. const int element=100;
4. class Stack
5. {
6.    int array[element];
7. public:
8.    Stack(){ for(int i=0;i<element;i++)
9.          array[i]=i;}
10.         ~Stack() { cout << "\n dieeeee\n";}
11.         getElement(int x)
```

```
12.         {
13.               if (x>100)
14.                     throw xThrow();
15.               else if (x<0) throw sThrow();
16.               else return array[x];
17.         }
18.         class xThrow{public:
19.         xThrow(){}
20.         xfalse(){   cout << "Big Element\n";}
21.         };
22.         class sThrow{
23.         public:
24.               sThrow(){}
25.               sfalse() {cout << "Small Element\n";}
26.         };
27.   };
28.
29.   int main()
30.   {
31.         int i=0; Stack a;
32.         for(;;){
33.         try{
34.         cout << "Choise The element\n";
35.         cin >> i;
36.         cout << "The element:\t" << a.getElement(i) << endl;
37.         }
38.         catch(Stack::sThrow a){
39.               a.sfalse();
40.         }
41.         catch(Stack::xThrow a){
42.         a.xfalse();
43.         }
44.         }
45.         return 0;
46.   }
```

- لا فرق في هذا المثال بينه وبين المثال السابق فستكون المخرجات هي نفسها ولكن الفرق بين الصنف Stack في هذا

المثال والمثال السابق هو أنه مستخدم الصنف Stack **في هذا المثال لن يهتم كثيراً بالتفاصيل الموجود في الصنف** Stack **فهي بسيطة للغاية الآن.**

- **انظر إلى تعريف الصنف** xThrow **تجد أنه يحتوي على تابعين الأول هو تابع بناء والآخر هو تابع يقوم بطباعة رسالة خطأ وهو** xfalse .
- **انظر إلى تعريف الصنف** sThrow **أيضاً تجد أنه يحتوي على تابعين الأول هو تابع بناء والآخر هو تابع يقوم بطباعة رسالة خطأ وهو** sfalse .
- **انظر أيضاً إلى كتل** catch **والأخطاء التي تقوم باصطيادها حيث أنها تقوم بإنشاء كائن من الصنف** sThrow **في السطر** 32 **والامر نفسه في كتلة** catch **الثانية في السطر** 41 .
- **تقوم هذه الكتلتين باستدعاء التابع** sfalse **و** xfalse **حسب الصنف الذي تم اصطياده.**
- **قد تقوم أنت بتطوير صنف وحيد يقوم بالتعامل مع جميع هذه الاستثناءات وقد تستخدم تعدد الاوجه بحيث لن يقلق المستخدم أبداً من احتمالية إلقاء استثناء غير صحيح.**

موضوع الاستثناءات موضوع كبير للغاية وشيق ولكن هذا الكتاب لن يقدم إلا أساسيات هذا الموضوع آملاً أن تكون عند حسن ظن مستخدميه.

التعامل مع الملفات

Handling With Files In C++

بداية:

كثيراً ما يستصعب مبتدئو البرمجة مواضيع التعامل مع الملفات ، والأمر ليس لصعوبة الموضوع بحد ذاته ، بل إلى طريقة عرضه والطريقة التي يحاول فيها المبتدئ التعامل مع الموضوع ، فهو ربما أنهى أصعب مواضيع البرمجة وخصوصاً تلك المواضيع التي تتصل بالذاكرة مثل المؤشرات ولربما قام بتنفيذ برامج كثيرة رائعة ، وربما في أحد الأيام أراد تطوير برنامجه ليكون قادراً على التعامل مع الملفات وحتى يفعل ذلك فإنه لا يأخذ هذا الموضوع بشكل جدي ويتجاوز أساسياته ليذهب بعيداً كي يتعامل مع المواضيع المتقدمة نسبياً والنتيجة لا شيء عدا إضاعة الوقت فيما لا يجدي ، وحتى تكون قادراً على فهم هذه الوحدة فأرجو منك أن تتعامل معها على أنها وحدة متكاملة لها أساسياتها الأولية وما إلى ذلك ؛ ولا تتعامل معها على أنها وحدة أمثلة تطبيقية فحسب.

العائلة ios:

قد تجد عنوان هذه الفقرة طريفاً ولكن بالفعل يوجد عائلة من الكائنات والأصناف هي العائلة ios **وقلما تجد برنامجاً في السي بلس بلس لا يستعمل هذه العائلة (ربما جميع البرامج) ، فهناك لديك الصنف الممتاز أو الصنف الأب** ios **الذي لدية ثلاثة أبناء ، الصنف** istream **و** ostream **والذين أورثا بالوراثة المتعددة خصائصهما إلى الصنف** iostream **، والابن الثالث هو** fsreampas **، بالنسبة لهذه الوحدة فهناك صنفان سنستخدمهما بكثرة الأول هو** ifstream **والصنف الثاني** ofstream**،والذين أخذا صفاتهما من الصنفين** istream **و** ostream **على التوالي .**

الملف Formatted File I/O:

لم أجد لمسمى formatted **مصطلحاً في اللغة العربية إلا أن الكتب العربية تطلق عليه الملفات النصية وعلى هذا الاسم سيتعامل معه الكتاب.**

في الملفات النصية تخزن المعلومات على شكل أحرف ، حتى الأرقام تصبح أحرفاً عند التخزين ، فإذا افترضنا أن برنامجك قام بتخزين الرقم 12.5 **من النوع** float **في أحد الملفات فإنه لن يخزن على هذا الشكل** 12.5 **أو حتى على أساس أنه رقم وليس حرف بل سيخزن هكذا** '1' '2' **و** '.' **و** '5' **أي** 4 **بايت حسب عدد الأحرف وليس حسب نوع البيانات، في بعض الأحيان يكون هذا جيداً للغاية وفي بعض ليس له أي داع أو أن بعض البرامج لن تنفذ بواسطة الملفات النصية.**

CODE

```
#include <iostream>
```

```
#include <fstream>
#include <string>
using namespace std;

int main()
      {
      char x = 's';
      int d = 77;
      double i = 3.14;
      string String1 = "Hellow";
      string String2 = "World";

      ofstream fout("data.txt");
      fout << x
        << d
        << ' '
        << i
        << String1
        << ' '
        << String2;
   cout << "File Completed\n";
   return 0;
   }
```

هذا الملف يقوم بتخزين جميع المتغيرات في البرنامج في ملف خاص نصي تحت مسمى data.txt ، كما تلاحظ فلقد أعلنا عن أحد الكائنات وهو fout من الصنف ofstream ، وقمنا بتمرير اسم الملف إليه ، حينما تفعل ذلك فسيقوم البرنامج بإنشاء ملف تحت نفس المسمى وسيقوم بكتابة تلك المعطيات أو المتغيرات إليه وبالطبع فهو يقوم بعملية الإخراج بواسطة معامل الإخراج << ، تحت نفس طريقة إستخدامه لنا بواسطة الكائن cout إلا أنه هذه المرة نتعامل مع الملفات وليس مع تيار الإخراج القياسي ، هناك ملاحظة جديرة بالإهتمام ألا وهي أننا قمنا بطباعة مسافة بين كل متغير ومتغير ، هذه ليست لتحسين الإخراج داخل الملف ، بل حينما نقوم بكتابة برنامج يقوم بقراءة المعلومات من الملف ، فإن البرنامج يستطيع التمييز بين كل متغير ومتغير .
سنقوم الآن بكتابة البرنامج الذي يقوم بقراءة هذه المعلومات من الملف data :

CODE

```
1. #include <fstream>
2. #include <iostream>
```

```
3. #include <string>
4. using namespace std;

5. int main()
6. {
7. char m;
8. int i;
9. double j;
10.      string chara;
11.      string chara2;

12.      ifstream fin("data.txt");

13.      fin >> m >> i >> j >> chara >> chara2;

14.      cout << m << endl
15.      << i << endl
16.      << j << endl
17.      << chara << endl
18.      << chara2 << endl;
19.      return 0;
20.      }
```

أول ما تلاحظه في هذا الكود أن جميع أسماء المتغيرات تغيرت ، والكود يقوم بفتح نفس الملف الذي قام الكود السابق بإنشاءه وفتحه ولكن هذه المرة تم تعريف صنف من نوع آخر وهو ifstream **هذا النوع من الكائنات يستخدم لقراءة وليس لكتابة المعلومات من الملفات ، كما تلاحظ فلقد قام الكائن** fin **بقراءة المعلومات وقام بتخزينها في أسماء المتغيرات المكتوبة ، لاحظ أن جميع أسماء المتغيرات تغيرت عن الكود السابق ولكن الأنماط لم تتغير ليس هذا لأن البرنامج يقوم بتخزين أسماء المتغيرات فهو لا يقوم بذلك أصلاً ولكن لأن كل ما يهتم به الكود أو البرنامج هو توافق المعلومات مع نوع المتغير في الملف المقروء ، بالنسبة للأعداد فكان بإمكانك تخزينها على أساس أنها أحرف.**

التعامل مع السلاسل:

لا تستطيع الأكواد السابقة التعامل مع السلاسل التي تنتهي بالرمز "\n"**، والسبب في ذلك أننا نستخدم الكائن** cin **، الذي لا يستطيع التعامل مع هذا النوع من المحارف ، وحتى نستطيع التعامل مع السلاسل المحرفية ينبغي علينا أن نستخدم الدالة** getline **، انظر لهذا الكود التالي:**

CODE

```
1. #include <fstream>
```

```
2. #include <iostream>
3. #include <string>
4. using namespace std;

5. int main()
6. {
7. ofstream fout("first.txt");
8. fout << "HELLOW EVREY ONE\n"
9.       << "You Are Here\n"
10.      << "in my program\n"
11.      ;
12.      return 0;
13.      }
```

يقوم البرنامج السابق بإنشاء ملف نصي هو first وتخزين ثلاث جمل أو سلاسل فيه في الأسطر 8و9و10 لاحظ أننا نفصل بينها بالمحرف "\n" وليس بالمسافة البيضاء ، الآن سنقوم بكتابة كود يتعامل مع هذه المحارف أو السلاسل ويقوم بتخزينها في سلسلة ثم يخرجها على الشاشة ، انظر لهذا الكود:

CODE

```
1. #include <fstream>
2. #include <iostream>
3. #include <string>
4. using namespace std;

5.  int main()
6.    {
7.     char Array[80];
8.     ifstream fin("first.txt");
9.     while ( !fin.eof() )
10.      {
11.      fin.getline(Array,80);
12.      cout << Array << endl;
13.      }
14.      ;
15.      return 0;
16.      }
```

أتينا في الكود السابق ببعض المفاهيم والدوال الجديدة ، يقوم السطر السابع بالإعلان عن المصفوفة التي ستقوم بتخزين السلاسل المحرفية

فـي الملـف first وفـي السـطر 8 ، تـم إنشـاء ملـف تحـت اسـم first ، ومـن الطبيعـي أن يجده البرنامج في نفس المجلد لأنك قمت بتنفيذ الكـود السابق وهذه المرة سيقوم بفتحـه للقراءة ولـيس للكتابـة عليـه فبالتـالي لـن يقـوم بحذف أي شيء من الملف ، في السـطر 9 يـدخل البرنـامج فـي دوارة while وهذه المرة فإن شرط هذه الدوارة غريب قليلاً فهـو الدالـة !fin.eof( ) ، وهـذه الدالة تتأكـد أن الملف انتهـى أو لـم ينتهـي ، يقـوم السـطر 11 بقـراءة الملـف عبر الدالة getline ويقوم بقراءة أي سلسلة حتى يصل إلى المحرف "\n" ثم يتوقـف عــن القـراءة ويقـوم بتخـزينها فـي السلسـلة Array ثـم يعـرض هـذه السلسلة على الشاشـة في السطر 12 ويستمر في العمـل هكذا حتى يصل إلى نهايـة الملف وبالتالي ينتهي تنفيذ البرنـامج هـكذا.

الدالة eof( ):

هذه الدالة تعـني نهايـة الملف وهـي إختصار لـ end of file ، مهـمـة هذه الدالة ، هذه الدالة تقوم بإرسال رسالة EOF (أي نهاية الملف) إلى البرنامج عبر نظام التشغيل حتى يتوقف البرنامج عـن القراءة ولا يستمر في القراءة إلى مالانهايـة .

الملفات الثنائيـة Binary File :

لا تستطيع الملفات النصيـة التعامل مع حالات أخــرى أكثـر تعقيداً مـن مجـرد نصوص ، فحينما نقوم بإنشاء أنظمـة أو برامج أكثر تعقيداً فإننا نحتـاج للتعامل معها على صـورتها الحقيقيـة ولـيس علـى أنها جميعها متغيـرات محرفيـة ، وهذا ما تقوم بـه الملفات الثنائيـة.

ونظراً لأن الملفات الثنائيـة تختلف عـن الملفات النصيـة فقد تجـد دوالاً أخــرى هـنا تختلف عـن الملفات النصيـة مع عـدم الإختلاف في الأساسيات.

هـذا النــوع مـن الملفـات لا ينظر للبيانـات علـى أنها نصـوص بـل علـى أنهـا متغيرات float و int وحتى أصناف تقوم بكتابتها أنت ، وهــناك أمـر آخر وهــو أن هذا النـوع من الملفات لا يهـتم كثيراً بكيفيـة تخـزين هذه البيانات ، كل ما يطلبه هـذا النــوع مـن الملفـات أن تحــدد حجـم البيانـات التـي تريد تخـزينها وحسب.

دوال جـديدة:

هـناك بعض الدوال الجـديدة مثل الدالة write والتي تقوم بالكتابة على الملف وهـي أحد أعضاء الصنف ofstream والدالة read( ) والتي تقوم بإستدعاء البيانات من الملف وقراءتها وهـي أحـد الدوال الأعضاء للكائن ifstream.

سنقوم الآن بكتابة كودين اثنين الأول يقوم بكتابة البيانات إلى ملف والآخر يقوم بقراءتها منه ، لا تقلق من هذه الأمثلة فحينما نصل إلى مواضيع متقدمـة أكثر سيكون هـناك كـود واحد يقوم بالقراءة والكتابة في آن معاً.

CODE

```
1. include <iostream>
2. #include <fstream>
3. using namespace std;
4. void main()
5. {
```

```
6. int Array[40];
7. int j=0;

8. for(int i=0;i<40;i++)
9. {
10.      j=i*10;
11.      Array[i]=j;
12.      }

13.      for (i=0;i<40;i++)
14.      {
15.      cout << Array[i] << endl;
16.      }
17.      ofstream fout("test.dat", ios::binary);

18.      fout.write (reinterpret_cast<char*>(Array),40*sizeof(int)
   );
19.      fout.close();
20.      }
```

يقوم هذا الكود بإنشاء مصفوفة مكونة من 40 عنصر ، ويخزن في كل عنصر في المصفوفة عدد يساوي رقم العنصر مضروب في 10 ، كما هو واضح من الكود ، المهم الآن هو السطر 17 يتم إنشاء أحد الملفات للإخراج وكما ترى فإن هذا الكائن (كائن من الصنف ofstream) يستقبل في دالة بناءه بارامترين اثنين الأول هو اسم الملف والبارامتر الثاني هو طريقة إنشاء هذا الملف ففي الوضع الإفتراضي سيكون هذا الملف نصي ولكن حينما نوضح طريقة الإنشاء فسيتم إستبعاد الطريقة الإفتراضية ووضع الطريقة التي كتبناها نحن وهي هكذا ios::binary . يقوم السطر 18 بكتابة جميع محتويات المصفوفة Array في الملف test.dat عن طريقة الدالة العضو write .

بارامترات الدالة write:

كما ترى في الكود السابق فإن الدالة write( ) تستقبل بارمترين اثنين سنناقشهما حالاً.

البارامتر الأول يتوقع مؤشر إلى متغير حرفي لذلك فيجب عليك تغيير نوع المصفوفة Array مؤقتاً ؛ أما البارامتر الثاني فهو حجم المصفوفة بالرغم من طول السطر 18 وتعقيده فهناك طريقة افضل من هذا التعقيد وهي كتابة السطر التالي بدلاً عن السطر 18 ، وكلا الطريقتين صحيحتين:

```
.          fout.write ( (char*) &Array, sizeof Array );
```

بالنسبة للسطر 19 فهو يقوم بإغلاق الملف وبالرغم من عدم ضرورة هذا الإجراء وخاصة في هذا الكود إلا أنه من الضروري أن تعود نفسك على إغلاق أي ملف حالما تنتهي منه ، لأن هذا يزيد من أمان برنامجك ولا يجعلك تكد في البحث عن أخطاء غبية.

سنقوم الآن بقراءة البيانات من الملف test وهذه المرة سنقوم بقراءتها بكل بساطة ونخزنها في مصفوفة أكبر حجماً ، وسيكون حجمها 60 عنصر وليس 40 وبالتالي فيجب علينا أن نحدد حجم المصفوفة التي نود القراءة منها في الملف وهي 40 من النوع int وليس 60 من النوع int كما هو في هذه الحالة ، تنبه لهذه النقطة جيداً:

CODE

```
1. #include <iostream>
2. #include <fstream>
3. using namespace std;
4.
5. void main()
6. {
7.       int Array[60];
8.       int i=0;
9.
10.      ifstream fin("test.dat", ios::binary);
11.      fin.read( (char*) &Array, 40*sizeof(int) );
12.
13.      for(i=40;i<60;i++)
14.           Array[i]=i*10;
15.
16.      for(i=0;i<60;i++)
17.           cout <<Array[i]<<endl;
18.
19.      fin.close();
20.      }
```

يقوم السطر 10 بفتح الملف للقراءة منه وليس للكتابة عليه فيما يقوم السطر 11 بالقراءة من الملف وتخزين البيانات في المصفوفة Array ، لاحظ الدالة read وخاصة في البارامتر الثاني ، لم نكتب الجملة التالي sizeof Array ، لأنك كما تعلم فإن حجم المصفوفة Array في هذا الكود هي 60 من النوع int ، والمخزن في الملف حسب الكود السابق هي 40 من النوع int ، لذلك قمنا بتحديد ما سيقرأه الكود وهي 40 من النوع int أي بالتحديد 160 بايت ، تقوم السطران 16 و 17 بإضافة العناصر العشرين المتبقية للمصفوفة Array ويقوم السطران 16 و 17 بطباعة المصفوفة بالكامل حتى تتأكد أن القراءة من الملف تمت بشكل صحيح وأن الإضافة للمصفوفة Array كانت صحيحة ؛ وفي النهاية يقوم السطر 19 بإغلاق الملف.

التعامل مع الأصناف والكائنـات:

لن نسـتفيد مـن الملفـات مـالم نتعامـل مـع الأصـناف ولـيس فقـط المتغيـرات لوحدها.
ولا يختلف التعامل مع الكائنـات من خلال الملفات بشـيء عـما تم ذكره سابقاً ولكن يجب علينا أن نذكر الطريقـة حتى يتم فهـم الموضوع بشكل جيد.

CODE

```
1. #include <iostream>
2. #include <fstream>
3. using namespace std;
4.
5. class Stud
6. {
7.        char itsname[100];
8.        int itsmark;
9.
10.       public:
11.             Data()
12.             {
13.                   cout << "Enter his name: " ;
14.                   cin >> itsname;
15.                   cout << "Enter his mark: " ;
16.                   cin >> itsmark;
17.             }
18.       };
19.
20.       void main()
21.       {
22.             Stud Ahmed;
23.             Ahmed.Data();
24.             ofstream fout ("Student.dat",ios::binary);
25.             fout.write((char*) &Ahmed, sizeof Ahmed);
26.             fout.close();
27.       }
```

قم بدارسة الكـود السابق مع العلم أنـه لا يوجـد أي شـيء مختلف عـما سبق ، فهـناك الصنف Stud ، له إحدى الدوال التي تسمح لمستخـدم الصنف بإدخال بيانات الكائن وفي السطر 25 يتم تخـزين بيانات هذا الكائن ، الآن سنقوم بقراءة البيانات من هذا الكائن ولكن بطريقـة أخـرى ، أنظر لهذا الكـود:

CODE

```
1. #include <iostream>
2. #include <fstream>
3. using namespace std;
4.
5. class Student
6. {
7.    char itsname[100];
8.    int itsmark;
9.
10.   public:
11.         DisplayData()
12.         {
13.               cout << "his name: " ;
14.               cout << itsname << endl;
15.               cout << "his mark: " ;
16.               cout << itsmark << endl;
17.         }
18.   };
19.
20.   void main()
21.   {
22.         Student Saeed;
23.         ifstream fin ("Student.dat",ios::binary);
24.         fin.read((char*) &Saeed, sizeof Saeed);
25.         Saeed.DisplayData();
26.         fin.close();
27.   }
```

كما ترى فلم يختلف الكود الحالي عن الكود السابق سوى في بعض الأسماء ، ولكن هذا لا يهم لأن حجم البيانات واحد في الاثنين وكما تعلم فإن السطر 24 يقوم بكتابة بيانات الملف student.dat وتخزينها في الكائن Saeed .

ملاحظات ضرورية للغاية:

تعاملنا عند كتابة الملف student مع الكائن stud وفي المرة الأخرى عند القراءة تعاملنا مع الكائن student ، وقد اختلفت أسماء الدوال في الكائنين بالرغم من صحة هذا الإجراء إلا أنه يعتبر خطير نوعاً ما وخاصة حينما نتعامل مع الكائنات المتوارثة والتي تحوي الدوال الظاهرية إن أي إختلاف في اسم الدالة الظاهرية أو الكائن المتوارث ظاهرياً سيؤدي إلى تغيير في البيانات المخزنة وبالتالي فشل القراءة من الملف لذلك احرص دائماً على

أن تكون الأسماء هي نفسها والكائنات هي نفسها ، بالنسبة للكود السابق فالسبب في نجاح الترجمة والتنفيذ هو كون المثال بسيط غير معقد وأيضاً أن الحجم هو نفسه وأن البيانات متوافقة مع بعضها البعض أي أنها مرتبة بنفس الترتيب ففي الكائن stud كان اسم الطالب هو أول البيانات لو كان اسم الطالب في الكائن student ليس أول البيانات فلربما سيختلف الوضع وبالتالي تفشل القراءة وتظهر لك أرقام غريبة جداً ، بل إن الأمر أكثر تطرفاً عند القراءة فلو كان أسلوبك أنت مختلف عن أسلوبي وقمت بكتابة البيانات العامة للكائن student أولاً ثم البيانات الخاصة لفشلت القراءة أيضاً ، لذلك يجب أن تكون البيانات متوافقة 100 % وحتى الأسماء سواء أسماء الكائنات أو أسماء الأعضاء.

التعامل مع الملفات والكائنات بطريقة أكثر تقدماً:

سنقوم الآن بالتعامل مع الملفات عن طريق كود واحد بدل كودين اثنين واحد للكتابة والآخر للقراءة ، وربما نتطور أكثر حتى نصل إلى طريقة تمكننا من التعديل على البيانات المخزنة والإستعلام عنها والكتابة فوقها أي ربما نصل إلى قاعدة بيانات لبرنامجنا، انظر لهذا الكود والذي يدمج الكودين السابقين في كود واحد فحسب:

CODE

```
1. #include <iostream>
2. #include <fstream>
3. using namespace std;
4.
5. class Student
6. {
7.
8.      char itsname[100];
9.      int itsmark;
10.
11.
12.     public:
13.           GetData()
14.           {
15.                 cout << "Enter his name: " ;
16.                 cin >> itsname;
17.                 cout << "Enter his mark: " ;
18.                 cin >> itsmark;
19.           }
20.           DisplayData()
21.           {
22.                 cout << "his name: " ;
23.                 cout << itsname << endl;
24.                 cout << "his mark: " ;
```

```
25.                  cout << itsmark << endl;
26.            }
27.
28.      };
29.
30.      void main()
31.      {
32.         char ch;
33.         Student Ahmed;
34.         fstream file;
35.
36.         file.open("Student.DAT", ios::app | ios::out |
37.                                  ios::in | ios::binary );
38.         do
39.            {
40.            cout << "\nEnter The Data Of Student:\n";
41.            Ahmed.GetData();
42.
43.            file.write( (char*) &Ahmed, sizeof Ahmed );
44.            cout << "Enter another person (y/n)? ";
45.            cin >> ch;
46.            }
47.         while(ch=='y');
48.
49.         file.seekg(0);
50.
51.         file.read((char*) &Ahmed, sizeof Ahmed );
52.         while( !file.eof() )
53.            {
54.            cout << "\nStudent:";
55.            Ahmed.DisplayData();
56.            file.read( (char*) &Ahmed, sizeof Ahmed );
57.            }
58.         cout << endl;
59.      }
```

لقد قمنا الآن بجعل الكائنين في الكودين السابقين كائناً واحداً في الكود الحالي وأطلقنا عليه اسم Student ، وبالطبع فلا جديد في هذا الصنف ، المهم هو أننا قمنا بالإعلان عن كائن file من الصنف fstream في السطر

34، وبالتالي فالآن بإمكاننا فتح أي ملف عبر هذا الكائن للقراءة والكتابة وكما ترى فإن الصنف fstream تستطيع التعامل معه على أنه صنف ifstream أو ofstream .

الدالة open( ) :

هذه الدالة هي أحد أعضاء الصنف fstream ، وهذه الدالة تستطيع فتح أي ملف لك من نفس الكائن ففي السابق كنا حتى نستطيع فتح ملف آخر علينا الإعلان عن كائن جديد ولا نستطيع فتح ملف آخر بواسطة نفس الكائن ، وهذه الدالة تقدم لك هذه الخدمة المفيدة بالإضافة لخدمات أخرى ، انظر إلى السطر 36 تجد أن الدالة open ، أخذت بارامترين اثنين ، البارامتر الأول هو اسم الملف والبارامتر الثاني هو طريقة فتح الملف وقد وجدت أربعة تعابير الأول هو ios::app ، وهي تعني الكتابة من نهاية الملف وليس من أوله أما التعبير الثاني فهو ios::out وهي تعني فتح الملف للكتابة (وهي الوضع الإفتراضي لكائنات الصنف ofstream) ؛ أما التعبير الثالث فهو ios::int وهي تعني فتح الملف للكتابة (وهي الوضع الإفتراضي لكائنات الصنف ifstream)؛ أما التعبير الرابع فهو ios::binary وهو يعني فتح الملف كملف ثنائي ؛ هذه التعابير الأربع يطلق عليها The Mode Bits والترجمة العربية لها حسب اعتقادي هي أنماط البتات ، أي كيف يتم فتح هذا الملف وقراءة وتخزين وكتابة البتات .

الوسيط	العمل الذي يقوم به
ios::app	يلحق الإدخال بنهاية الملف
ios::ate	يقوم بالقراءة أو الكتابة من نهاية الملف
ios::trunce	في حال وجود الملف فيسقوم ببترها أي حذف محتوياتها
ios::noceate	إذا كان الملف غير موجود تفشل عملية الفتح
ios::noreplace	إذا كان الملف موجود تفشل عملية الفتح
ios::in	فتح الملف للقراءة(حالة افتراضية لكائنات ifstream)
ios::out	فتح الملف للكتابة (حالة افتراضية لكائنات ofstream)
ios::binary	فتح الملفات على هيئة ثنائية وليس كنصية.

السطر 43:

يتم كتابة كائنات الصنف Student في الملف من خلال هذا السطر إلا أن الأمر الجيد أن الكتابة تبدأ في نهاية الملف وليس من أوله ، لأن الكتابة إذا ابتدأت من أول الملف فسيضطر البرنامج إلى مسح البيانات السابقة وكتابة بيانات جديدة عليها وبالتالي ضياع بيانات الكائنات الأخرى.

بقية الكود :

الدالة seekg() أحد الدوال الأعضاء للصنف fstream ، وهي تقوم بالتحكم في مؤشر الكتابة أو القراءة من الملف ، فهي تستطيع تحديد من أين يتم القراءة والكتابة من الملف ولقد وضعنا لها القيمة 0 حتى تبدأ القراءة أو الكتابة من أول الملف وليس من آخره في السطر49 ، وبالتالي فحينما تتم القراءة من في السطر 51 فإنها لا تتم لآخر كائن مخزن بل إلى أول كائن مخزن في الملف وحينها يدخل البرنامج في تنفيذ الدوارة while والتي تشترط الوصول إلى نهاية الملف حتى تتوقف عن الدوران. وفي نهاية تنفيذ كل دورة من دوارة while في السطر 56 فإنها تقرأ الكائن التالي حتى تصل إلى نهاية الدوارة while وبالتالي الخروج من البرنامج.

التنقل داخل الملفات File Pointers:

نحن هنا نتحدث عن كيفية تحديد موقع مؤشر القراءة أو الكتابة داخل الملف ، لكل ملف متغيرين من النوع int الأول هو Put Pointer أي قم بوضع مؤشر القراءة أو الكتابة في مكان معين والثاني هو get pointer أي حدد المكان الفعلي لهذه المؤشرة.

ويطلق على هذين المتغيرين اختصاراً اسم: the current position.

وبالطبع تحدد هذين المتغيرين موقع البايت داخل الملف الذي ستبدأ القراءة منه والكتابة.

هناك دالتان سنقوم بإستخدامهما هما : seekg( ) و tellg( ) .

دعنا نتحدث قليلاً عن الدالة Seekg( ) ؛ لهذه الدالة حالتي إستخدام الأولى ببارامتر واحد والثانية ببارمترين اثنين ، بالنسبة للحالة الأولى فهي تأخذ بارامتر واحد هو رقم البايت التي تود وضع المؤشرة للبداية منه ، لقد رأينا في الكود السابق أننا وضعنا الرقم 0 وبالتالي فهي تبدأ من البايت رقم صفر والذي هو بداية الملف ؛ ماذا لو وضعت الرقم 104 وهذا الرقم هو حجم الصنف Student لقام البرنامج بطباعة الطالب الثاني المخزن في الملف ولن ينتبه للطالب الأول ، ماذا لو وضعت الرقم 2 ، هل تعرف ما هو الرقم 2 ، البايت رقم 2 هو الحرف الثاني من اسم الطالب إذا وضعت هذا الرقم فستظهر أشياء غريبة للغاية لذلك أنصحك بعد تجربة الطريقة ؛ بالنسبة للحالة الثانية من حالات إستعمال الدالة seekg( ) هي بارامترين اثنين البارامتر الثاني هو نمط بتات من ثلاثة أنماط هي ios::beg أي بداية الملف والثانية ios::cur أي موقع المؤشرة الحالي والثالثة هي ios::end هي نهاية الملف ، أما بالنسبة للبارامتر الأول فهو نفسه ولكن دلالته تتغير فالآن يصبح يحدد موقع البايت ليس من بداية الملف ولكن من البارمتر الثاني فالتعبير:

```
.            fin.seekg(-50,ios::end);
```

يعني أن يذهب المؤشر إلى البايت رقم 50 قبل نهاية الملف ، ونفس الأمر ينطبق على بقية الأنماط الثلاثة.

بالنسبة للدالة tellg( ) ؛ فهي تقوم بإعادة رقم البايت الذي يشير إليه المؤشرة حالياً في الملف.

الآن وبعد أن قمنا بشرح هذه الدالتين المهمتين عند التنقل ضمن الملفات سنقوم الآن بكتابة كود يبحث في الملف Student حسب ترتيب هذا الطالب ضمن الملف. قم بدارسة الكود التالي قبل شرحه وتذكر أنه بسيط للغاية.

CODE

```
1. #include <fstream>
2. #include <iostream>
3. using namespace std;
4.
5. class Student
6. {
7.
8.       char itsname[100];
```

```
9.      int itsmark;
10.
11.
12.     public:
13.             GetData()
14.             {
15.                     cout << "Enter his name: " ;
16.                     cin >> itsname;
17.                     cout << "Enter his mark: " ;
18.                     cin >> itsmark;
19.             }
20.             DisplayData()
21.             {
22.                     cout << "his name: " ;
23.                     cout << itsname << endl;
24.                     cout << "his mark: " ;
25.                     cout << itsmark << endl;
26.             }
27.
28.     };
29.
30.     void main()
31.     {
32.     Student Ahmed;
33.     ifstream looking;
34.     looking.open("Student.dat",ios::in || ios::binary);
35.
36.     looking.seekg(0,ios::end);
37.
38.     int endF=looking.tellg();
39.     int x= endF /sizeof(Ahmed);
40.     cout << "There are " << x << " Student:\n";
41.
42.     cout << "Enter the number of student you want to display
   his data\n";
43.     cin >> x;
44.     int position = (x-1) * sizeof(Ahmed);
45.     looking.seekg(position);
```

```
46.
47.        looking.read( (char*) &Ahmed, sizeof Ahmed);
48.        Ahmed.DisplayData();
49.        }
```

بالرغم من طول هذا الكود إلا أنه بسيط للغاية والفكرة فيه هي أن يقوم أولاً بحسب عدد بايتات الملف ثم يقسمه على عدد بتات حجم الصنف Stuudent وسيكون الناتج هو عدد الكائنات المخزنة في الملف أو عدد الطلاب والفكرة الثانية هي أنه يطلب من المستخدم إدخال رقم الطالب الذي يريد الإستعلام عنه ولنفرض أنه رقم 3 سيقوم البرنامج بضرب الرقم 3 في حجم الصنف Student ثم يبدأ من بداية الملف ويحرك المؤشرة حسب عدد البايتات الناتجة من عملية الضرب السابقة ويصل إلى الطالب المعني ويقوم بطباعة بياناته.
في السطر 33 تم الإعلان عن الكائن looking من النوع fstream وتم فتح الملف للقراءة في السطر 34 ، في السطر 36 قام البرنامج بتحريك المؤشرة إلى نهاية الملف ثم طلب من الدالة tellg( ) تحديد رقم البايت الأخير في الملف وبالتالي نستطيع معرفة حجم الملف في السطر 38 وفي السطر 39 نعرف عدد الطلاب الحقيقي في الملف حسب الفكرة السابقة.
يقوم المستخدم بإدخال رقم الطالب الذي يود الإستعلام عنه في السطر 43 وفي السطر 44 يتم تحديد رقم البايت الذي يبدأ فيه تخزين الطالب المحدد وفي السطر 45 تنتقل المؤشرة إلى الطالب ليقوم البرنامج بعد ذلك بقراءة الطالب وطباعة بياناته.
أعتذر عن الشرح السريع ولكن الكود بسيط للغاية ولا يمنع من أن تطيل التفكير فيه قليلاً.

كيف تجعل الكائنات أكثر تماسكاً:

الهدف من البرمجة الشيئية هو جعل كائناتك أكثر تماسكاً وتعليمات إستخدامها بسيطة للغاية وفي هذه المرة عند التعامل مع الملفات فسيكون هناك الكثير من التعليمات حينما تريد من مستخدم الصنف حفظ البيانات ، وحتى تجعل الكائنات أو الأصناف أكثر تماسكاً فعليك إضافة بعض الدالات إليها ، ومن هذه الدالات دالة لحفظ البيانات ودالة لقراءتها.

تضمين أوامر التعامل مع الملفات داخل الأصناف:

سنقوم الآن بإعادة إنشاء الصنف Student حتى نضمن دالات التعامل مع الملفات بداخله وليس خارجاً من الدالة main( ) ؛ بإمكان هذا الكود رغم إختلافه التعامل مع الملف Student ، الذي أنشأته الأكواد السابقة:

CODE

```
1. #include <fstream>
2. #include <iostream>
3. using namespace std;
4.
5. class Student
6. {
7.
8.        char itsname[100];
```

```
9.      int itsmark;
10.
11.
12.     public:
13.             GetData()
14.             {
15.                     cout << "Enter his name: " ;
16.                     cin >> itsname;
17.                     cout << "Enter his mark: " ;
18.                     cin >> itsmark;
19.             }
20.             DisplayData()
21.             {
22.                     cout << "his name: " ;
23.                     cout << itsname << endl;
24.                     cout << "his mark: " ;
25.                     cout << itsmark << endl;
26.             }
27.             DataIn(int x)
28.             {
29.                     ifstream infile;
30.         infile.open("Student.dat", ios::binary);
31.         infile.seekg( x*sizeof(Student) );
32.         infile.read( (char*)this, sizeof(*this) );
33.             }
34.             DataOut()
35.             {
36.                     ofstream outfile;
37.
38.         outfile.open("Student.dat", ios::app | ios::binary);
39.         outfile.write( (char*)this, sizeof(*this) );
40.             }
41.             static int ManyInFile()
42.             {
43.                     ifstream infile;
44.         infile.open("Student.dat", ios::binary);
45.         infile.seekg(0, ios::end);
46.         return (int)infile.tellg() / sizeof(Student);
```

```
47.             }
48.
49.       };
50.
51.       int main()
52.          {
53.          Student Ahmed;
54.          char x;
55.
56.          do {
57.             cout << "Enter data for Student:\n";
58.             Ahmed.GetData();
59.             Ahmed.DataOut();
60.             cout << "Register another (y/n)? ";
61.             cin >> x;
62.             }  while(x=='y');
63.
64.          int n = Student::ManyInFile();
65.          cout << "There are " << n << " persons in file\n";
66.          for(int j=0; j<n; j++)
67.             {
68.             cout << "\Student " << j;
69.             Ahmed.DataIn(j);
70.             Ahmed.DisplayData();
71.             }
72.          cout << endl;
73.          return 0;
74.          }
```

بقي أن أشير هنا في نهاية هذا الموضوع إلى أن ما ذكرناه هو فقط مقدمة لكيفية تنظيم الملفات والبحث عن الكائنات فمثلاً لو أردنا منك أن تقوم بالبحث عن الطلاب الذين تزيد درجتهم عن 80 وتعديلها إلى 81 ، فإن الامثلة السابقة لا تنفع ويبقى عليك أنت البحث عن طريقة وأنصحك بالبحث عنها في كتب الخوارزميات وتراكيب البيانات ويفضل أن تكون بلغة السي بلس بلس وليس بلغة أخرى لأنك ستضيع إن قمت بدراستها بلغة أخرى وحاولت تطبيقها على السي بلس بلس.

الأخطاء عند استعمال الملفات:

قد تحدث حينما يقوم البرنامج بالتعامل مع الملفات بعض الأخطاء والتي يجب أن تكون قادراً على التعامل معها.

حسناً ، لنفرض أن لدينا ملف اسمـه Test.dat ، وقـد قمنا بكتابـة كــود يقـوم بفتح هذا الملف ، والقراءة منه وفي حال عـدم وجود هذا الملف فسيتم قراءة بيانات افتراضية غير الملف ، وسيلتنا لتحقيق هذه الغايـة هــي عبـر if . انظـر إلى هذا المثال:

```
1. ifstream file;
2.     file.open(“a:test.dat”);
3.
4.     if( !file )
5.         cout << "\nCan’t open test.DAT";
6.     else
7.         cout << "\nFile opened successfully.";
```

تقوم الحلقة if باختبار وجود الملف test ، وفي حال عـدم وجوده ينطلق تقوم بتنفيذ السطر 5 وفي حال وجوده تقوم بتنفيذ السطر 7 .
وفرت لك هذه الفقرة كيفية التحقق من وجود ملف ما ، ويبقى لك أنت القيـام بكتابة تطبيقات لمشاريعـك بهذه التقنيـة.

مكتبة القوالب القياسية

Standard Template Library

بداية:

جميع الشركات الكبرى أصبحت تقدم الآن مكتبة القوالب القياسية STD ضمن مترجماتها ، وهذه المكتبة تقدم لك خدمات كبيرة للغاية وقد استخدمنا بعضاً من خدماتها في مكتبتي iostream و string ، توفر لك هذه المكتبة المتجهات وهي بديل أفضل من المصفوفات والمؤشرات وأيضاً توفر لك أنواع عديدة من القوائم المرتبطة إضافة إلى بعض التوابع التي تقوم بخوارزميات البحث والفرز إلخ.

تستخدم هذه المكتبة مساحة الأسماء العامة std .

وأيضاً تستخدم القوالب ، وقد أجلت الحديث عن هذه الوحدة حتى الخوض في موضوع القوالب حتى تكون أيسر للفهم.

محتويات هذه المكتبات:

تحوي هذه المكتبات أنواع عديدة وكثيرة من الخدمات والكائنات إلا أن هناك ثلاثة أشياء تحويها هي الأهم ؛ وهي: الحاويات ، الخوارزميات ، أما عن الثالثة فلم أجد لها مثيل في اللغة العربية بالرغم من بحثي المتواصل لمعناها وهي Iterators ، أعتقد أن معناها هي التكرارات ولكن نظراً لأني لم أجد لها المصطلح المقابل لها بالعربية فسأعاملها كمصطلح إنجليزي.

مقدمة إلى الحاويات:

الحاويات هي طريقة لتخزين البيانات في الذاكرة وتنظيمها، هناك بعض الحاويات التي تعاملنا معها وهي المصفوفات والقوائم المترابطة ، هناك أيضاً أنواع وأشكال من القوائم المترابطة لم نذكرها وأيضاً هناك حاويات تشبه المصفوفات إلا أنها تتفوق عليها ، وعموماً تقسم الحاويات هنا إلى نوعين: الحاوية التسلسلية وهي مثل المصفوفات حيث بإمكانك الوصول إلى العنصر الذي تريده من خلال الفهرس ، والنوع الثاني هو الحاوية الترابطية وهي مثل القوائم المترابطة ، حيث لا وجود للفهرس وإنما فقط يتم البحث من خلال قيم محددة داخل هذه الحاوية وتنظيم هذه الحاوية ليس مثل المصفوفات بل مثل القائمة المترابطة.

كائنات التكرار Iterators :

كائنات التكرار هي تعميم لمفهوم المؤشرات وهي تشير إلى العناصر الموجودة في الحاويات ، في القوائم المترابطة لا وجود لمعامل الفهرس كما هو الحال في المصفوفات والمتجهات لذلك يجب عليك استخدام كائنات التكرار. وسنستعرضها هذه التقنية ضمن الحاويات.

نظرة عامة إلى الحاويات:

يحوي هذا الجدول أهم الحاويات الموجودة في المكتبات القياسية:

الحاوية	الوصف	المميزات
vector	مصفوفة أحادية البعد . بالإمكان تغيير حجمها متى شئت	تستخدم الفهرس للوصول إلى عناصرها بطيئة عند وضع عناصر أو مسح عناصر في المنتصف سريعة عند وضع عناصر أو مسحها في الأطراف
list	قائمة مرتبطة زوجياً	وصول عشوائي بطيء وضع العناصر ومسحها وحذفها والوصول إليها أسرع
deque	طابور	وصول عشوائي سريع بطيئة عند المسح أو الإضافة في المنتصف. سريعة عند الحذف أو الإضافة في الأطراف
queue	طابور	كالسابق
stack	رصات	تستخدم كغلاف للحاويات. يزداد حجمها ويتقلص من الطرف الخلفي فقط. لا يمكن الوصول إلى عناصرها أو محوها إلا من الطرف الخلفي
map	مصفوفة ترابطية	يمكن الوصول إلى العناصر بمفتاح واحد فقط
set	مجموعة	يمكن الوصول إلى العناصر بمفتاح واحد فقط.

هناك أيضاً عدد آخر من الحاويات ولكن تعرضنا هنا لأهمها.

تحوي هذه الحاويات أيضاً بعض التوابع المشتركة بينها ، هل تتذكر الكائن string ، أغلب التوابع الأعضاء الموجودة فيه يكاد يكون معظمها موجوداً هنا.

التابع	العمل
begin( )	يعيد هذا التابع العنصر الأول في الحاوية
end( )	يعيد هذا التابع العنصر الأخير في الحاوية
front( )	الوصول إلى أول عنصر
back( )	الوصول إلى آخر عنصر
push_back( )	إضافة عنصر من الطرف الخلفي للحاوية
pop_back( )	حذف آخر عنصر من الطرف الحلفي للحاوية
push_front( )	إضافة عنصر من الطرف الأمامي للحاوية
pop_front( )	حذف عنصر من الطرف الأمامي

	للحاوية
insert(a , b)	ضع العنصر b قبل العنصر a في الحاوية
insert(a,b,c)	أضف العنصر c بعدد b نسخة قبل العنصر a
insert(a,first,last)	أصف العناصر من first إلى last قبل العنصر a حيث الوسيطان الثالث والرابع يتبعان لحاوية أخرى
erase(a)	احذف العنصر رقم a من الحاوية
erase(a,b)	احذف العناصر من a إلى b من الحاوية
clear( )	احذف جميع عناصر الحاوية
size( )	يعيد هذا التابع عدد عناصر الحاوية
empty( )	يعيد هذا التابع القيمة1 إذا كانت الحاوية خالية
capacity( )	عدد العناصر التي تستطيع الحاوية استيعابها قبل تخصيص وإعادة تخصيص للذاكرة
swap(a,b)	بدل العنصر a مكان العنصر b مع العلم أنهما من حاويتين مختلفتين
find(m)	اعثر على رقم العنصر m

المتجهات vector :

سنتعرف الآن على إحدى أقوى الحاويات وهي المتجهات ، المتجه شبيه بالمصفوفة العادية لدرجة عالية ، إلا أن المميز في المتجهات هو قدرتها على تغيير حجمها متى أردت فعل ذلك .
ليست المتجهات مثل المصفوفة الديناميكية ، في المصفوفة الديناميكية يجب على المستخدم تحديد حجم المصفوفة في أحد أوقات تنفيذ البرنامج أما في المتجهات فلا يشترط أصلاً أن تقوم بتحديد أي حجم للمتجه ، سنقوم الآن بكتابة مثال عملي يوضح لك أهم خصائص ومميزات المتجهات حاول أن تستطيع فهمه حتى تستطيع فهم بقية الحاويات:

CODE

```
1. #include <iostream>
2. #include <vector>
3. using namespace std;
4.
5. int main()
6. {
7.    vector <double> v;
8.    double k=0;
9.
10.         cout <<"please enter your grade in all course?"
11.               << " (for out pree 0)\n";
```

```
12.
13.         for (int i=1;;i++){
14.               cout << "please enter your grade in course" << i
15.                     << ":\t";
16.               cin >> k;
17.               if (k==0)break;
18.               v.push_back(k);
19.         }
20.
21.         int j=v.size();
22.         double total=0 ,avg=0;
23.         for (i=0;i<j;i++)
24.               total+=v[i];
25.
26.         avg=total/j;
27.
28.         cout << endl << endl;
29.         cout << "Your total of grades is:\t\t" << total << endl;
30.         cout << "Your Avg is\t\t\t\t" << avg << endl;
31.
32.         return 0;
33.   }
```

- هـذا البرنـامج يقـوم بحسـاب درجـات الطالـب ويوجـد مجموعهـا والمعدل لها. في السطر الثاني قمنا بتضمين المكتبة vector حتى نقوم بتخزين درجات الطالب فيها.
- في السطر 7 ، تم استخدام الكائن vector لاحظ كيف أن اسـتخدام هذا الكـائن شـبيه باسـتخدام قوالـب الكائنـات ، وقـد جعلنا جميـع محتويات هـذه الحاوية عبـارة عـن الـنمط double ، المتغيـر v هـو المتجـه الذي سيضم درجات الطالب.
- يدخل البرنامج في حلقة for أبديـة ، ثم يطلـب منـه إدخـال درجـة المادة في السطر 16 وفي حال أدخل العـدد 0 ، يخرج البرنامج من الحلقة for وينتهي إدخال الدرجات ، أما إذا أدخـل المسـتخدم رقمـاً آخر فسيتم تخزينه كعـنصر جديـد في المتجـه vector.
- في السطر 18 يتم دفع العـنصر الذي أدخلـه المسـتخدم إلـى ذيـل المتجـه v عبر التابع push_back( ) .
- في السطر 21 وبعـد الخروج من الحلقة for يحسب البرنـامج عـدد المـواد التـي أدخلها الطالـب والتـي هـي فـي هـذه الحالـة حجـم المتجـه وسيتم حسابه عبر التابع size( ) .
- يتم حساب مجمـوع الدرجات في الأسطر 23 و 24 ، لاحظ هـنا أننا قمنا باستخدام معامل الفهرس [ ] .
- يتم حساب معـدل الطالب في السطر 26 .

- يطبع البرنامج جميع هذه المعلومات في السطرين 29 و 30 .

الآن عد إلى وحدة الصنف string وقم بتطبيق التوابع الأعضاء فيها على المتجهات.

كما قلنا أن المتجهات لا تعمل إلا من طرف واحد هو الطرف الخلفي وبالتالي فلن يكون بإمكانك استخدام التابع push_front( ) .

القوائم Lists :

هناك أيضاً نوع آخر غير المتجهات وهي list ، قبل استخدامها يجب تضمين ملف الرأس list .
يجب علينا هاهنا استخدام كائنات التكرار للتأشير لأن القوائم المترابطة تعتمد على المؤشرات وليس على الفهارس كما في المصفوفات والمتجهات ، سنقوم الآن بتعديل المثال السابق حتى نستطيع هنا استخدام القوائم:

CODE

```
1. #include <iostream>
2. #include <list>
3. using namespace std;
4.
5. typedef list<int> grade;
6.
7. int main()
8. {
9.    grade v;
10.         double k=0;
11.
12.         cout <<"please enter your grade in all course?"
13.               << " (for out pree 0)\n";
14.
15.         for (int i=1;;i++){
16.               cout << "please enter your grade in course" << i
17.                     << ":\t";
18.               cin >> k;
19.               if (k==0)break;
20.               v.push_back(k);
21.         }
22.
23.         int j=v.size();
24.         double total=0 ,avg=0;
25.   for (grade::const_iterator ci = v.begin();
```

```
26.                                                             ci != v.end(); ++ci)
27.                  total+=(*ci);
28.            avg=total/j;
29.
30.            cout << endl << endl;
31.            cout << "Your total of grades is:\t\t" << total << endl;
32.            cout << "Your Avg is\t\t\t\t" << avg << endl;
33.
34.            return 0;
35.      }
```

- لا مميز إلى الآن في هذا المثال عن المثال السابق سوى استخدامنا القوائم بدلاً من المتجهات وأيضاً وجود السطر 5 حيث أصبح بإمكانك حالياً التعامل مع المسمى grade وكأنه نمط جديد من البيانات.
- لا غريب في هذا المثال إلا حينما نستخدم كائنات التكرار وبالتحديد في السطر 25 و 26 .
- كما قلنا أن كائنات التكرار عبارة عن تعميم لمفهوم المؤشرات كما ترى فلقد استخدامنا أحد كائنات التكرار الثابتة وهذا يدل على أننا لا نريد تغيير العقدة أو القائمة بشكل عام ، إذا ما نظرت جيداً لهذا الجزء من السطر 25 :

```
grade::const_iterator ci = v.begin();
```

- فستجد أنه هو نفسه هكذا:

```
list<int>::const_iterator ci = v.begin();
```

- تلاحظ هنا أن كائن التكرار عبارة عن كائن معرف ضمن تعريف الصنف list ، هناك عدة كائنات للتكرار ولكننا هنا استخدمنا كائن التكرار const_iterator وقمنا بجعله يؤشر إلى أول عنصر في القائمة لاحظ أن هذا الكائن ثابت أي لا يتغير .
- انظر إلى شرط الحلقة for في السطر 26 تجد أنه يطلب من كائن التكرار ci عدم التأشير إلى نهاية القائمة list ، لاحظ أيضاً الزيادة في الحلقة for تجد أنه يجعل المؤشر يشير إلى العنصر التالي في القائمة.
- ألاحظ أيضاً السطر 27 حيث يقوم هنا بحسب المجموع total .
- ألا تذكرك العلاقة بين كائنات التكرار والحاويات بنفس العلاقة التي بين المصفوفات والمؤشرات.
- لا يوجد أي شيء آخر غريب في هذا الكود عن الكود السابق.

الحاوية deque :

لا فرق بينها وبين الحاويات الأخرى ، يكمن الفرق بينها وبين أي حاوية أخرى هي المميزات التي تمتاز بها هذه الحاوية عن غيرها ، لاحظ هنا أنه لا فرق بين أي حاوية عن حاوية من ناحية الواجهات (التوابع وكائنات التكرار والأعضاء) عدا شيء قليل لا يذكر ، لكن يكمن الفرق في مميزاتها فقط ، وحسبما تريد أنت استخدامه في برنامجك.

بعض التوابع الأعضاء الآخرين:

سنتعرف الآن على تابعين آخرين هما ( )merge و ( )reverse وسنرى فائدتهما في هذا المثال:

CODE

```
1. #include <iostream>
2. #include <list>
3. using namespace std;
4.
5. int main()
6.    {
7.    int j,i;
8.    list<int> list1, list2;
9.
10.      for(j=0,i=0; j<4; j++,i=j*2)
11.         list1.push_back( i );    //list1: 0, 2, 4, 6
12.      for(j=0,i=0; j<5; j++,i=j*3)
13.         list2.push_back( i );    //list2: 0, 3, 6, 9, 12
14.      cout << "list1:\t";
15.
16.      for(list<int>::const_iterator c=list1.begin();
   c!=list1.end();
17.      ++c)
18.            cout << *c << "\t";
19.      cout << endl;
20.      cout << "list2:\t";
21.
22.      for(list<int>::const_iterator a=list2.begin();
   a!=list2.end();
23.      ++a)
24.            cout <<  *a <<"\t";
25.      cout << endl;
26.
27.      list1.reverse();                  // list1: 6 4 2 1
28.      list1.merge(list2);               //list1+=list2
29.
30.
31.      int size = list1.size();
```

```
32.      while( !list1.empty() )
33.         {
34.         cout << list1.front() << ' ';
35.         list1.pop_front();
36.         }
37.      cout << endl;
38.      return 0;
39.      }
```

- هناك قائمتين هما list1 و list2 وأعضاء هاتين القائمتين مكتوبتين في السطرين 11 و 13 على التوالي ، تقوم الأسطر 16 – 25 بطباعة محتويات هاتين القائمتين.
- في السطر 27 وعبر التابع reverse( ) يتم عكس ترتيب هذه القائمة وهذه هي وظيفة التابع .
- في السطر 28 وعبر التابع merge( ) يتم دمج القائمة list1 في القائمة list2 .
- الحلقة while المعرفة في الأسطر 32 إلى 35 تقوم بطباعة محتويات القائمة list1 ، حيث يقوم السطر 34 بطباعة العنصر الأول في القائمة ، ثم يقوم السطر 35 بإلقاء أو إخراج العنصر الاول من القائمة خارجاً ثم تستمر الحلقة بالدوران حتى تصبح القائمة فارغة وينتهي البرنامج.

الحاويات الترابطية Associative Containers :

تعرفنا على ثلاثة أنواع من الحاويات هي vector و deque و list ، هذه الحاويات ترتب فيها البيانات وتخزن على شكل مصفوفة حيث يمكن الوصول السريع إليها ، يدعى هذا النوع من الحاويات بالحاويات التسلسلية نظراً لكونها مثل السلسلة (سلسلة بطرفين).

أما النوع الآخر من الحاويات فهو الحاويات الترابطية ، حيث لا تخزن البيانات بشكل مرتب أو بشكل مفهرس كالسلسلة ، بل هي مرتبة بشكل أكثر تعقيداً شبيه بالأشجار وليس بالسلسلة ، حيث أن البيانات لا تخزن على أساس مفهرس بل على أساس قيمها ، أقرب مثال لذلك هو كشف طلاب الفصل ، حيث أن هذا الكشف لا يرتب على أساس أول طالب مسجل بل على أساس قيمة أساسية وهي الترتيب الألفبائي ، الآن لو أتى طالب جديد فلن نقوم بضمه إلى آخر كشف الطلاب ولا حتى لأوله بل حسب ترتيبه الألفبائي الذي قد يكون في المنتصف أو في أي مكان آخر ، بالطبع فإن الحاويات الترابطية أكثر تعقيداً من هذا المثال ، فهي في الأساس لا تخزن مثل كشف الطلاب بجانب بعضها البعض ، بل على شكل مبعثر ، ولكن كل عنصر يرتبط ليس بعنصر واحد فقط بل بأكثر من عنصر (عنصرين في الغالب) بواسطة المؤشرات .

البحث في هذه الحاويات ليس بواسطة الفهرس بل بواسطة القيم الأساسية أي لو كان لدينا قاعدة بيانات للموظفين فلن يتم البحث فيها حسب رقم الموظف بل يمكن إن أردنا البحث فيها على أساس اسم الموظف أو عمره أو أي شيء آخر.

لذلك فإن الحاويات المترابطة هي أفضل وأسرع في الترتيب والبحث ولكنها أكثر إنهاكاً للمترجم.

هـناك أربع حاويات ترابطية هـي set و multset و map و multmap .
الحاويـة set تقـوم بتخـزين الكائنـات علـى أسـاس امتلاكهـا مفتاحـاً أو قيمـة أساسية كالاسم مثلاً أما الحاوبة map فتقوم بتخـزين زوجاً حيث الـزوج الاول عبارة عـن كائن يحوي مفتاحاً والزوج الثاني يحوي قيمـة.عموماً لا تقلـق فـي حال عدم فهـمك آلية عـمل كلاً منهـما فسنصل إلى ذلك عـما قريب.

الحاوية set :

تستخدم الحاوية set كقاعدة بيانات لك في حال ما أردت استخدامها للأصناف التـي تقـوم أنـت بكتابتهـا أو صـناعتها ، بـالطبع لـيس هــناك فـي الحاويـات المترابطة التوابع push و pop لأنه لا وجود للعـنصر الأول ولا العـنصر النهـائي فيها.
الطريقة المثلى لوضع العناصر في هذه الحاوية هي عبر التابع العضو insert ، أيضـاً لا ننســى أن علينا هــنا اسـتخدام كائنـات التكـرار ولـيس الفهـرس ، سنقوم الآن بكتابة كـود نقوم فيه بتخزين قائمـة أسماء لأشـخاص ، نسـتطيع البحث فيها وإضافة أشخاص آخرين أيضاً ، أنظر إلى هذا المثال الكـودي:

CODE

```
1. #include <iostream>
2. #include <set>
3. #include <string>
4. using namespace std;
5.
6. int main()
7. {
8.    set <string> names;
9.          names.insert("Mohamed");
10.         names.insert("Ahmed");
11.         names.insert("Sultan");
12.         names.insert("Emad");
13.         names.insert("Thamier");
14.
15.         string a;
16.         set<string>::const_iterator i;
17.
18.         for (i=names.begin();i!=names.end();++i)
19.               cout << *i << endl;
20.         char sure;
21.         for(;;){
22.         cout <<  "\nDo you want to add another(y/n):\n";
23.         cin >> sure;
24.
25.
```

```
26.          if(sure=='y'){
27.                cin >> a;
28.                names.insert(a);}
29.          else if (sure=='n') break;
30.          else cout << "Try againe\n";
31.          }
32.
33.          for(;;)
34.          {
35.                cout << "Do you want to find a name\n";
36.                cin >> sure;
37.                if(sure=='y'){
38.                cout << "Enter the name\t";
39.                cin >> a;
40.                i=names.find(a);
41.                if ( i== names.end()) cout << "Not in there\n";
42.                else cout << "we found it\n";
43.                }
44.                else if (sure=='n') break;
45.                else cout << "try againe please\n";
46.          }
47.          cout << endl << "Think for using this\n";
48.
49.          return 0;
50.    }
```

- في السطر الثاني قمنا بتضمين المكتبة set حتى نستطيع استخدام كائناتها وتخزين البيانات التي نريدها.
- في السطر 8 قمنا بإنشاء كائن مجموعة set وهو names .
- في الأسطر 9 -18 وعبر التابع العضو insert قمنا بإضافة 5 أعضاء من الصنف string إلى الحاوية names .
- في السطر 15 قمنا بالإعلان عن كائن string حتى نستخدمه لأغراض البحث ، وفي السطر 16 أعلنا عن كائن تكرار ليقوم هو i حتى نستغله في طباعة عناصر الحاوية.
- السطرين 18 و 19 وعبر حلقة for يقوم البرنامج بطباعة عناصر الحاوية بنفس الآلية التي شرحناها في الأمثلة السابقة.
- سنمكن المستخدم من إدخال أسماء جديدة قدر ما يشاء ووسيلتنا إلى ذلك هي حلقة for الأبدية التي يدخل فيها البرنامج في السطر 21 .
- يطلب السطر 23 من المستخدم إدخال أحد اختيارين هما y أو n ، إذا ما أراد إدخال اسم جديد أو لأ ، في حال اختار حرفاً آخر فسيتم

تنبيهه إلى ذلك في السطر 30 وإعادة حلقة for لنفسها وإعادة الطلب مرة أخرى.

- في السطر 29 يخرج البرنامج من حلقة for إذا ما أدخل المستخدم الحرف n.
- إذا اختار المستخدم حرف y لإضافة أسماء جديدة ، فسيطلب منه إدخال سلسلة حرفية للكائن a في السطر 27 ثم توضع في هذه السلسلة في الحاوية names في السطر 28 .
- يدخل البرنامج في حلقة for أبدية أخرى والسبب في ذلك هو إمكانية أن يقوم بالبحث في هذه القائمة قدر ما يشاء وذلك في السطر 33 .
- بنفس الآلية السابقة فلن يتم البحث إلا إذا اختار المستخدم الحرف y ، وفي حال اختاره فسيطلب منه في السطر 39 إدخال الاسم الذي يريد البحث عنه .
- في السطر 40 يستدعى تابع البحث find ليبحث عن الاسم الذي أدخله المستخدم وستسند القيمة التي يعود بها إلى كائن التكرار i .
- السطر 41 يتأكد إن كان البرنامج وجد السلسلة المطلوبة أو لا ، حيث يتم مقارنة كائن التكرار i بالقيمة المعادة من التابع العضو end وفي حال كانتا متساويتان فإن هذا يعني عدم وجود السلسلة أما في حال عدم المساواة فهذا يعني وجودها وبالتالي طباعة جملة للمستخدم بذلك في السطر 42 .
- السطر 47 يطبع رسالة توديعية لمستخدم البرنامج.

الخريطة map :

الخريطة تقوم بتخزين زوج من الكائنات الأول هو عبارة عن كائن مفتاحي والكائن الثاني هو عبارة عن قيمة لهذا الكائن المفتاحي ، إن الأمر أشبه ما يكون بمصفوفة ترابطية تتألف من بعدين البعد الأول عبارة عن مثلاً كائنات string والبعد الثاني عبارة عن درجات لعناصر البعد الأول أو أرقام حساب لعناصر البعد الأول أو أي شيء آخر ، أنظر إلى هذا المثال حتى تفهم هذا الكلام النظري:

CODE

```
1. #include <iostream>
2. #include <map>
3. #include <string>
4. using namespace std;
5.
6. int main()
7. {
8.    string a;double x;
9.    string name[]={"Ahmed","Iman","Amani","Mohamed","Fadi"};
10.         double numOfTel[]={12548,15879,13648,14785,5826};
11.
12.         map<string, double> mapTel;
13.      map<string, double>::iterator i;
```

```
14.
15.      for(int j=0; j<5; j++)
16.         {
17.         a = name[j];
18.         x = numOfTel[j];
19.         mapTel[a] = x;
20.         }
21.
22.      cout << "Enter name: ";
23.      cin >> a;
24.      x = mapTel[a];
25.      cout << "Number_Of_Tel: " << x << "\n";
26.
27.      cout << endl;
28.      for(i = mapTel.begin(); i != mapTel.end(); i++)
29.         cout << (*i).first << ' ' << (*i).second << "\n";
30.      return 0;
31.      }
```

لقد قمنا في هذا الكــود بإنشــاء دليـل للهواتـف بطريقــة بسيطــة للغاية ، لا يعتقد منها أن تكون معقدة.
في السطر الثاني قمنا بتضمين محتويات المكتبة map .
في السطـريـن التاسـع والعاشـر أعلنا عــن مصـفوفة أسـماء string أما فـي السطـر العاشر فقد أعلنا عـن مصفوفة أرقام (أرقام هاتف).
في السطـر 12 قمنا بوضع المصفوفتين السابقتين فـي حاويـة map واحـدة ، انظر إلى هذا السطر:

```
map<string, int> mapTel;
```

تجـد أن الوسيط الأول هـو المفتاح أو مفتاح الوصول للنتائج التي تريدها ، كما تعلم ففي دليـل الهـاتف النـاس يبحثـون بواسطــة أسـماء الأشـخاص لإيجـاد أرقام هواتفهـم ولا يبحثون بواسطـة أرقام الهـواتف لإيجاد أسماء الأشخاص ، لذلك فسيكون الوسيط الثاني هـو القيمـة والتي هي أرقام الهاتف في هـذه الحالة.
في السطـر 13 قمنا بالإعلان عـن كائن تكرار هـو i على نفس نسـق الحاويـة في السطـر 12.
في الأسطـر مـن 15-20 يتم وضـع العــناصر أو المصـفوفتين السـابقتين فـي الحاوية.
في السطـر 23 يطلب منك البرنامج إدخال اسم للبحث عـنه خـلال الخريطــة.
يتم وضع الاسم الذي تبحث عـنه بين قوسين فهرس في الخريطــة وإذا وجـد البرنامج الاسم في الحاوية فسيعيد رقـم هاتفـه إلى المتغير x وفي حال لـم يجده أصلاً فسيعيد القيمـة 0 إلى المتغير x في السـطـر 24 . وسـتتم طباعــة قيمـة المتغير x في السطـر 25.
في السطرين 28 و 29 ستتم طباعـة جميع محتويات الخريطـة map .

إذا قمت بإدخال اسم ليس موجوداً في الخريطة فستتم إضافته كعنصر جديد إلى الخريطة وبإمكانك إضافة قيمة إليه أي رقم هاتف بواسطة كائن ومعامل الإدخال.

الخوارزميات Algorithms :

توفر لك مكتبات STL بعض التوابع التي تقدم لك خدمات شاملة للحاويات، من فرز وبحث ودمج واستبدال وعد وغير ذلك .
التوابع الموجودة عبارة عن قوالب لذلك فبإمكانك استخدامها على حاويات STL أو على حاويات قمت أنت بكتابتها أو حتى على المصفوفات العادية.
سنتعرف في هذه الفقرة على أهم الخوارزميات وتذكر أن مكتبة القوالب القياسية في السي بلس بلس أشمل من أن يشملها هذا الكتاب ، بسبب كبر حجمها ومميزات الخدمات التي تقدمها للمبرمجين.
في الحقيقة ليست الخوارزميات عبارة عن توابع بل هي بشكل أوضح عبارة عن كائن تابع ، وكائن التابع هو عبارة عن كائن لا يحوي سوى على تابع لزيادة تحميل المعامل ( ) ، سنتعرف الآن على هذا المثال ، حيث سنقوم الآن بكتابة كائن تابع شبيه بالتابع القوي في لغة السي printf ، حيث سنجعله بشكل مبدئي يطبع قيمة واحدة فقط.

CODE

```
1. #include <iostream>
2. #include <string>
3. using namespace std;
4.
5. template <class T>
6. class prin {
7.    public:
8.          void operator() (const T& t)
9.          {
10.                     cout << t ;
11.               }
12.   };
13.   prin <string> print;
14.
15.   int main()
16.   {
17.         string a;
18.         cin >> a;
19.         print(a);
20.       print("\n");
21.
22.         return 0;
23.   }
```

لقد قمنا الآن بإنشاء كائن نتعامل معه على أنه تابع عادي ، وبإمكاننا الآن استخدامه ولو بشكل مبسط كالتابع printf بالرغم من الفروق الواضحة جداً بينهما.
بنفس شاكلة هذا التابع print تمت كتابة الخوارزميات فالخوارزميات في الأساس هي عبارة عن كائنات نتعامل معها على أنها توابع وليست توابع بحد ذاتها.الذي أقصده هنا أن التوابع التابعة للمكتبة algorithm في أغلبها توابع أما التوابع التابعة للمكتبة functional فهي كائنات توابع.

خوارزمية البحث find( ) :
تستخدم خوارزمية البحث للبحث عن قيمة محددة ، وتأخذ هذه الخوارزمية ثلاث وسائط ، الوسيط الأول هو الحاوية التي تود البحث عنها والتي قد تكون مصفوفة ، والوسيط الثاني هو إلى أي عنصر تود أن يستمر البحث ، والوسيط الثالث هو العنصر التي تود إيجاده ، انظر إلى هذا المثال الذي يبحث في مصفوفة عددية من النوع int :

CODE

```
1. #include <iostream>
2. #include <algorithm>
3. using namespace std;
4.
5. int main()
6. {
7.     int number[]={1,5,8,10,85,100,89};
8.     int a;
9.
10.                cout << "Enter the number\n";
11.          cin >> a;
12.          int* num=find(number,number+7,a);
13.          cout << "The number in\t" << (num-number) << endl;
14.
15.          return 0;
16.
17.    }
```

- تم تضمين المكتبة algorithm في السطر الثاني.
- يطلب البرنامج من المستخدم إدخال العدد الذي يود البحث عنه في السطر 11.
- يتم البحث عن العدد الذي أدخله المستخدم في السطر 12 بواسطة التابع find ، حيث أن التابع find يستقبل أولاً اسم الحاوية والبارامتر الثاني هو حجم الحاوية أو عدد الأعضاء الذي سيتم البحث فيهم وفي البارامتر الثالث يتم وضع القيمة التي تود البحث عنها خلال الحاوية ، التابع find يعيد مؤشر وليس متغير.
- في السطر 13 يتم طباعة رقم العنصر الذي وجد فيه العنصر.

خوارزميـة الترتيب ( )sort :

يستقبل التابع sort بارامترين اثنـين فقـط ، البـارامتر الأول هــو اســم الحاويـة التي تود وضعه ، والبارامتر الثاني هـو حجم الحاويـة . انظر إلى هذا المثال:

CODE

```
1. #include <iostream>
2. #include <algorithm>
3. using namespace std;
4.
5.
6. int main()
7. {
8.    int number[]={1,45,80,40,-1,60,55};
9.
10.         for(int a=0;a<7;a++)
11.               cout << number[a] << "\t";
12.
13.         cout << endl;
14.         cout << "The array after sorting\n";
15.         sort(number,number+7);
16.
17.         for( a=0;a<7;a++)
18.               cout << number[a] << "\t";
19.
20.         cout << endl;
21.         return 0;
22.
23.   }
```

أما ناتج البرنامج فهـو كالتالي:

```
1. 1  45    80    40    -1    60    55
2. The array after sorting
3. -1 1     40    45    55    60    80
```

لا يحتاج هذا المثال إلى شرح فهو واضح للغاية

خوارزميـة العـد ( )count :

هذه الخوارزميـة قريبـه من خوارزمية البحث إلا أن عـمـل هذه الخوارزميـة هـو عـد عدد مرات تكرار أحد العـناصر.

يستقبل هذا التابع ثلاث وسائط ، البارامتر الأول هـو اسـم الحاوية والبـارامتر الثاني هـو حجم الحاوية والبارامتر الثالث هـو العـنصر الذي تـود عـدـه ، انظـر إلى هذا المثال:

CODE

```
1. #include <iostream>
2. #include <algorithm>
3. using namespace std;
4.
5.
6. int main()
7. {
8.    int number[]={1,40,80,40,40,60,55};
9.
10.         for(int a=0;a<7;a++)
11.               cout << number[a] << "\t";
12.
13.         cout << endl;
14.         int n=count(number,number+7,40);
15.
16.               cout << "The times of 40 is:\n" << n << endl;
17.
18.         return 0;
19.
20.   }
```

وناتج هذا البرنامج هـو كالتالي:

```
      40     80     40     40     60     55
      The times of 40 is:
      3
```

لاحظ أن عـدد مرات تكـرار العـنصر 40 هـي ثلاث مـرات التـابع count يقـوم بحساب عـدد مرات التكرار وبالتالي فإن الناتج هـو 3 مرات.

خوارزمية لكل من for_each( ) :

تنفذ هذه الخوارزمية أحد التوابع على جميع أعضاء حاوية مـا، تسـتقبل هـذه الخوارزمية ثلاث بـارامترات ، البـارامتر الأول هـو أول عـنـصر والبـارامتر الثـاني هـو العـنصر آخر عـنصر والبارامتر الثالث هـو التـابع الـذي تـود تطبيقة علـى جميع عـناصر الحاوية ابتداءً من البارامتر الأول إلـى البـارامتر الثـاني ، عـمـوماً أنظر إلى هذا المثال:

CODE

```
1. #include <iostream>
2. #include <algorithm>
3. using namespace std;
4.
5. template <class T>
6. class prin {
7.    public:
8.          void operator() (const T& t)
9.          {
10.                     cout << t ;
11.               }
12.   };
13.   prin <int> print;
14.
15.
16.   int main()
17.   {
18.         int   Int[]={1,2,3,4,5};
19.
20.
21.         cout << "for_each()\n";
22.
23.         for_each(Int, Int+5, print);
24.
25.         cout << endl;
26.
27.         return 0;
28.   }
```

قمنا بكتابة كائن تابع هـو prin وهـو نفسه الذي قمنا بكتابته في مثال سابق من هذه الوحدة.في الأسطر من 5 إلى 12.
في السطر 13 قمنا بتعريف هذا الكائن.
يبـدأ عـمل for_each فـي السـطر 23 ، حيـث سـتقوم بتمريـر أول عضـو مـن المصفوفة وحتى آخر عضو إلى الكائن print والذي سيقوم بطباعتها وهـكذا تخلصنا للأبد من تعقيد for حينما نريد طباعة أعضاء عـناصر حاوية ما.

لقد تعرضنا فقط لإحدى أهـم الخوارزميات ولم نتعرض في هـذه الوحـدة إلـى كائنـات التوابع والمكتبة functional **، لأن الهـدف من هذه الوحدة هو تعريفـك بقدرة المكتبات القياسية للسـي بلس بلس على العـطاء.**

مثال عملي

Program Example

بدايـة:

لقد كـان فـي نيتـي أن أجعـل المثـال الأخيـر فـي هـذا الكتـاب شـاملاً لجميـع المفاهيم التي تناولها الكتـاب ، أقصـد هـنا نـواحي البرمجـة الكائنيـة ، أيضـاً أردتـه في نفس الوقت مثالاً يشرح فيه الكتـاب التصـميم الموجـه للكائنـات ، كمثال الآلة الحاسـبة أو نظـام ATM ، إلا أن وقـت تـأليف هـذا الكتـاب جعلنـي أسارع في كتابة مثال بسـيط ولكنـه شـامل لأغلـب مواضيع الكتـاب ولـيس جميعها وهـو حاوية بسيطـة

مثال/

سنقوم في هذا المثـال بكتابة حاويـة تسلسـلية بسيطـة للغاية لها بعـض المميزات ، إلا أننا لن نصل إلى حاوية خارقـة بل إلى حاوية تناسـب الأغـراض التعليميـة لهـذا الكتـاب ، الحاويـة شبيهــة للغاية بالمتجهــات ، وتحـل أيضـاً مشاكل المصـفوفات ، وفيها أيضاً بعـض المميـزات الجـديدة التـي لا تملكهـا المتجهـات ، إلا أن عـملياتها الداخلية لتخزين البيانات لن تكون خارقـة كما هـو الحال في المتجهـات.

الحل:

سنقوم بكتـابة هذا البرنـامج هـكذا:

في البداية وقبل كـل شـيء علينا أن نعلـم شـيئاً قبـل كتابة أي شــيء فـي البرمجـة ألا وهـو أن علينا أن نكتب هذا الصنف أو الحاوية بحيـث تكـون قـادرة علـى خدمــة جميـع المسـتخدمين ولـيس نحـن فقـط ، أيضـاً يجـب أن نحـدد الواجهـة لهذا الصنف وألا نجعل العـمليات الداخلية واجهـة.

الآن علينـا أن نحـدد مسـؤوليات الصـنف أو الحاويـة التـي نريـد كتابتهـا ، أي بمعــنى أصح هـل نجعـل هـذه المهمــة مـن مهـام المسـتخدم الـذي يريـد استعـمال الحاوية أو من مهام الحاوية ، أي هل هذه المهام ستكون من مهام الحاوية أو مهام العـناصر التي ستحتويها.

بعـد أن نكون الآن حددنا الواجهـة ومسؤوليات الصنف الذي نود إنشـائه ومـاذا يعمل ، نتساءل الآن حول كيفية عـمل الصنف ، أي ما هي العـمليات الداخلية وكيف سيتم حجز الذاكرة للعـناصر .

لقد حددنا نـوع التخزين لهذه الحاوية ، ألا وهـي حاويـة تسلسـلية ، قـد تقـول الآن أنك ستجعلها سلسلة من المؤشـرات التـي تشـير إلـى بعضـها الـبعض، الأمـر يعـود إليـك ولكـن فـي هـذا المثـال سنعـتمد علـى طريقـة المصـفوفة الديناميكية .

هذه الحاوية تقوم بحجز ذاكرة للعـناصر التـي قمـت بتهيئتهـا بهـا ، فـي حـال قمت بإضافة لهذه الحاوية فإنها ستسأل إن كـان هــناك مكـان إضـافي حتـى تضع العـنصر الجـديد وفـي حـال لـم يكــن هــناك فإنهـا ســتقوم بحجـز ذاكـرة جـديدة وتضع فيها الذاكرة القديمــة بالإضـافة إلــى العــناصر الجـديدة وتقـوم بإلغاء وحذف الذاكرة القديمـة.

بالإضافة إلى عـمليات الزيادة فبإمكان المستخدم أيضاً حذف أي عـنصر لا يريده من المصفوفة ، وعلينا الآن أن نفكر في كيفية فعل ذلك ، الوسيلة

الوحيدة حتى نستطيع حذف عـنصر من حاوية ما ، يمكن تشبيهها بأنك تقوم بسحب كتاب من مجموعـة كتب فوق بعضها البعض ، الذي سيحدث حينما تقوم بسحب الكتاب من المنتصف أن الكتب أعلاه سنسقط على المكان الذي سحبت منه الكتاب ولكن مجموعـة الكتب هذه لن تنهار أو تسقط وتتشتت على الأرض ، وهذا ما عليك فعله ، في هذه الحاوية الأمر شبيه بالرصات Stack .

بإمكان المستخدم أيضاً حفظ الحاوية في ملف أو حلب حاوية من نفس النوع من ملف ، وهـناك أيضاً بعض الإضافات.

إليك الآن إعلانات أعضاء هذه الحاوية:

CODE

```
1. template <class T>
2. class array
3. {
4.     int _size;
5.     int _capacity;
6.     T *arr;
7.     T *arr2;
8.     chapter(int m);
9.     void allocater();
10.         void allocater(int x);
11.         void alloce ();
12.         public:
13.                 array();
14.                 array(int m);
15.                 void save();// لوضع الحاوية في ملف ما
16.                 void load();//  لتحميل حاوية من ملف ما
17.                 int size();
18.                 int capacity();
19.                 void erase(int x);// لحذف عنصر من الحاوي
20.                 void push_back(T x);// لإضافة عناصر جديدة
21.                 void clean(); // لحذف جميع عناصر الحاوية
22.                 int find (T x)const;// للبحث داخل الحاوية
23.                 void operator()(int m);// لإعادة تخصيص الذاكرة للعناصر
24.                 array<T> operator+ ( array<T>& rhs);//لدمج حاويتين
25.                 array<T> &operator=(array<T> &rhs);
26.                 T &operator[](int x);// للوصول إلى عناصر الحاوية
27.     };
```

سنتعرف الآن على تعريف كل عضو من هذه الأعضاء وما يقوم به أو ماهي فائدتـه.

العضو _size :

كما ترى فإن هذا العضو ، وظيفتـه هـي إعلام مستخدم الصنف بالحجم الحالي للحاوية وعـدد عـناصرها فقط.

العـضو _capacity :

وظيفـة هذا العضو هـو إعلام المستخدم بالحجم الحقيقي للحاوية وليس عـدد عـناصرها ، وما هـو عـدد العـناصر الذي حينما تصل إليه الحاوية يتم تخصيص وإعادة تخصيص جـديد للذاكرة.

العضو المؤشر arr :

هذا العضو هـو المصفوفة الديناميكية التي ستضم جميع العـناصر التي ستقوم أنت بإضافتها

العضو المؤشر arr2 :

هذا العضو هـو مصفوفة ديناميكية أخرى لكن لا تظهر أي فائدة لها إلا حينما نرغب في تخصيص وإعادة تخصيص للذاكرة.

بقية الأعضاء المكبسلين:

هؤلاء التوابع تكمن فائدتهـم في عـملية المعالجة الداخلية وتخصيص وإعادة تخصيص الذاكرة للعـناصر ، وسنقوم بشرحهـم عـما قريب.ضمن هذه الوحدة.

تقسيم الذاكرة (التابع chapter) :

حينما تعاملنا مع المتجهات في وحدة سابقـة ، كنت تجـد أن تخصيص وإعـادة تخصيص الـذاكرة يـتم وفـق خوارزميـة أو آليـة معينـة وليسـت عشـوائيـة ، وظيفـة التابع chapter ، هـو أنه يقوم بأخذ عـدد العـناصر التي ستريد وضعها في الحاوية ثم يقوم بتحـديد الحجم المناسـب التـي سـتكون عليـه الحاوية ؛ هذا التابع لا يقوم بأي عـملية على الذاكرة ولكـن فقـط يقـوم بتحـديد الحجـم المناسب ، هذا التابع يقوم بتقسيم الذاكرة إلى 31 مرحلة ، كل مرحلـة تضـم 30 عـنصر فالمرحلة الأولى 30 عـنصر والمرحلـة الثانيـة 60 والمرحلـة الثالثـة 90 ، حتى يصل إلى المرحلة 31 وهـي 930 ، إذا كان عـدد العـناصر أكثر فـإن الحاوية تنهار ، بإمكانك أنت إضافة المزيد إذا أردت . لاحظ أن الطريقة المتبعـة فـي هـذا التقسـيم ليسـت طريقـة ينصـح بهـا بـل يفضـل أن تقـوم بجعلهـا خوارزميـة بدلاً من أن تكون طويلة للغاية كما فـي تعريـف هـذا التـابع ، ونظـراً لأن هذا التابع لن يقوم مستخدم الصنف باستخدامـه أبداً لأنـه مـن العــمليات الداخلية للصنف فسيكون مكبسلاً ، هذا هـو تعريف الصنف:

```
1. template <class T>
2. int array<T>::chapter(int m){
3.              if (m<0) throw;
4.              else if (m<30) return 30;
5.              else if (m<60) return 60;
6.              else if (m<90) return 90;
7.              else if (m<120)return 120;
8.              else if (m<150)return 150;
```

```
9.                  else if (m<180)return 180;
10.                       else if (m<210)return 210;
11.                       else if (m<240)return 240;
12.                       else if (m<270)return 270;
13.                       else if (m<300)return 300;
14.                       else if (m<330)return 330;
15.                       else if (m<360)return 360;
16.                       else if (m<390)return 390;
17.                       else if (m<420)return 420;
18.                       else if (m<450)return 450;
19.                       else if (m<480)return 480;
20.                       else if (m<510)return 510;
21.                       else if (m<540)return 540;
22.                       else if (m<570)return 570;
23.                       else if (m<600)return 600;
24.                       else if (m<630)return 630;
25.                       else if (m<660)return 660;
26.                       else if (m<690)return 690;
27.                       else if (m<720)return 720;
28.                       else if (m<750)return 750;
29.                       else if (m<780)return 780;
30.                       else if (m<810)return 810;
31.                       else if (m<840)return 840;
32.                       else if (m<870)return 870;
33.                       else if (m<900)return 900;
34.                       else if (m<930)return 930;
35.                       else throw;
36.                  }
```

تذكر لا وظيفة لهذا التابع سوى تحديد الحجم المناسب للذاكرة ، إذا لم تفهم المغزى من هذا التابع فعليك الاستمرار في قراءة هذه الوحدة حتى تصل إلى تطبيقات هذا التابع ضمن التوابع الأعضاء الآخرين.

تابع البناء : array

دعنا الآن نقوم بتحديد وظيفة هذا التابع ، هذا التابع يجب أن يكون أولاً مرناً في الاستخدام وثانياً عليه حجز حجم الذاكرة المناسب للعناصر في الحاوية ، بالنسبة لمرونة هذا التابع فبإمكان المستخدم حجز الذاكرة يدوياً بوضعه عدد العناصر التي يريدها أو أن يتم حجزها آلياً في حال نسي المستخدم ذلك ، أنظر إلى تابع البناء:

```
1. template <class T>
2. array<T>::array()
```

```
3.          {
4.                  _size=1;
5.                  _capacity=chapter(1);
6.                  arr=new T[_capacity];
7.                  arr[0]=0;
8.          }
```

هذه النسخة من التابع تفترض أن المستخدم لن يستعمل إلا عنصراً واحداً فقط أو أنه سيتعامل مع الحاوية على أنها تحوي عنصر واحد فقط.
كما ترى فلقد استخدمنا القوالب لأنها حاوية نريدها لجميع الأصناف والعناصر وليست فقط للعناصر التي نريدها.
أنظر إلى رأس التابع في السطرين 1 و 2 ولاحظ كيفية كتابة هذا التابع خارج تعريف الصنف array .
في السطر 4 يتم تحديد حجم عناصر الحاوية بأنها عنصر واحد فقط.
في السطر 5 تأتي فائدة التابع chapter ، حيث يقوم تابع البناء بإرسال عدد عناصر الحاوية وهو 1 كبارامتر إلى هذا التابع ، القيمة المعادة من هذا التابع هي 30 ، وسيتم إسنادها للمتغير _capacity .
في السطر 6 يتم حجز الذاكرة للمصفوفة الديناميكية ، لاحظ أن هذه المصفوفة قالب تقبل جميع الأصناف وليس صنفاً واحداً فحسب.
لو دققت النظر قليلاً فستجد أن هذه المصفوفة الديناميكية لن يتم حجز إلا عنصر وحيد لها وسيتم إسناد الصفر إليها في السطر 7 .
الآن دعنا نتعامل مع الحالة الأخرى وهي في حالة قام المستخدم بتحديد عدد العناصر التي يريدها.
الحل لذلك هو زيادة تحميل تابع البناء ، أنظر هاهنا :

```
1. template <class T>
2. array<T>::array(int m):_size(m),_capacity(0)
3.          {
4.    _capacity=chapter(m);
5.    int d=0;
6.    arr=new T[_capacity];
7.          for (int i=0;i<_capacity;i++)
8.                  arr[i]=d;
9.          }
```

أنظر إلى رأس التابع في السطرين 1 و 2 ، لاحظ أن عضو الحجم _size تمت تهيئته بالعدد الذي قام المستخدم بتمريره وهو عدد العناصر التي يريد حجزها في الحاوية ، أما المتغير الآخر وهو capacity ، فيتم تهيئته بالرقم 0 ، والسبب الوحيد لذلك هو إحدى أساليب البرمجة الآمنة وهي لا تدع متغيراً بدون أن تقوم بتهيئته.
في السطر الرابع يتم تحديد الحجم المناسب للذاكرة بواسطة التابع chapter ويقوم البرنامج بإسناد هذه القيمة إلى المتغير _capacity .
في السطر 6 يتم حجز الذاكرة للمؤشر arr ليس بعدد العناصر التي أرادها المستخدم ولكن بزيادة قليلة ، وقد تتساءل عن السبب أو الفائدة ، عموماً الفائدة هي حتى لا نزيد إنهاك المترجم ، فلو قرر المستخدم زيادة حجم الحاوية بعنصر وحيد فقط فلا سبيل لذلك إلا بإعادة تخصيص الذاكرة من

جديد ، أيضاً هذه الوسيلة أحد الحلول التي تقدمها لك هذه الحاوية عندما يخرج المستخدم خارج حدود الحجم.
السطران 7 و 8 يقومان بإسناد قيمة الصفر إلى جميع أعضاء الحاوية أو إلى جميع عناصر المصفوفة الديناميكية arr ؛ والسبب لفعل ذلك هو أمان الصنف فماذا لو قام المستخدم باستعمال أحد عناصر الحاوية التي نسي إسنادها بقيمة ما.

تابع الإضافة push_back :
هذا التابع مشهور للغاية وهو يقوم بدفع العناصر إلى الحاوية من الطرف الخلفي لها ، فلو افترضنا أنه يعمل في حالة المصفوفات العادية فهو لا يقوم بإضافة العناصر ضمن المصفوفة بل خارج حدود المصفوفة ، ويأتي هذا التابع كحل لمشاكل عديدة فهو يعفي المستخدم من مسؤولية السؤال كل ثانية عن حجم الحاوية ، ويمكنك من إضافة العناصر إلى الحاوية دون أن تتأكد من الحجم أو أي شيء آخر ، ولقد رأينا هذا التابع كثيراً في مكتبات القوالب القياسية ، أنظر إلى تعريف هذا التابع:

```
1. template <class T>
2. void array<T>::push_back(T x)
3.          {
4.                  if(_size+1<_capacity)
5.                  arr[++_size]=x;
6.                  else{ allocater();arr[++_size]=x;}
7.
8.          }
```

انظر إلى رأس التابع في السطرين 1 و 2 كما ترى فإن هذا التابع يستقبل العنصر الجديد الذي تريد إضافته إلى الحاوية.
السطر 4 يسأل الحاوية إن كانت غير مملوءة وفي حال كانت غير مليئة بالعناصر فإنه يقوم بزيادة العنصر _size زيادة واحدة فقط ، ويضيف عنصر الحاوية الجديد إلى ما بعد العنصر الأخير.
في حال كانت الحاوية مملوءة ولا تقبل أي عنصر آخر فبالتالي علينا هنا أن نتعامل مع مشكلة الذاكرة أي علينا تخصيص وإعادة تخصيص للمؤشرات ، لم نكلف هذا التابع بهذه المهمة فلقد جعلنا يقوم باستدعاء التابع allocater والذي يقوم بإعادة تخصيص الذاكرة ، لاحظ أن هذا التابع لا يقوم بزيادة عدد عناصر الحاوية (أي حجمها) وإنما يقوم بزيادة المساحة التخزينية للذاكرة ، لا تهتم بالتفاصيل الداخلية لهذا التابع فسأصل إلى شرحه حالاً ، في السطر 6 وبعد تخصيص وإعادة تخصيص الذاكرة يتم إضافة العنصر الجديد إلى الحاوية وزيادة عدد العناصر (أو الحجم) زيادة واحدة.

التابع allocater :
لهذا التابع نسختين أي أنه محمل ، النسخة الأولى تستقبل بارامتر واحد وهو عدد العناصر التي تريد تخصيص ذاكرة إليها والنسخة الثانية لا تستقبل بارامترات وإنما تقوم آلياً بزيادة الذاكرة ، سنتحدث أولاً عن النسخة الثانية بلا وسائط .

أنظر إلى تعريف هذا التابع:

```
1. template <class T>
2. void array<T>::allocater()
3.          {
4.                arr2=arr;
5.                _capacity+=30;
6.                arr=new T[_capacity];
7.                for(int i=0;i<_capacity-30;i++)
8.                      arr[i]=arr2[i];
9.                delete[] arr2;
10.                     arr2=0;
11.
12.               }
```

انظر إلى رأس التابع في السطرين 1 و 2.
كما قلنا سابقاً أن هـناك مصـفوفتان ديناميكيتـان ، الأولـى أساسـية والثانية احتياطية لا يتم حجز الذاكرة إليها إلا في حـال التخصـيص وإعـادة التخصـيص فقط.
في السطر 3 يتم نسـخ المصـفوفة الديناميكيـة الأساسـية arr ووضـع جميـع عــناصرها فــي المصــفوفة الاحتياطيــة . لاحــظ هــنا أن هــاذين المؤشــرين يشيران إلى نفس المصفوفة وبالتالي فأي حـدث الآن على أحدهـما سيكون لـه نفس الأثـر علـى المؤشـر الثـاني ، أي أن هـاذين المؤشـرين مـرتبطين ويستحيل الفصل بينهما بالطرق التقليدية.
فــي الســطر 5 يــتم رفــع الطاقــة الاســتيعابية للحاويــة أي زيــادة المتغيــر _capacity ثلاثين عـنصر والسبب (في كونها 30) هـو طريقة تقسيم الذاكرة الذي اعتمدناه منذ البداية قد تود اعتماد عـنصر آخر ولكن الآن نحن نتعامل مع هذه الطريقة.
بالرغـم من زيادتنا للمتغير _capacity إلا أن الذاكرة لم تزد بعـد.
في السطر 6 يتم فك الارتبـاط بـين المصـفوفتين الأساسـية arr والاحتياطـية arr2 ، من خلال حجز ذاكرة جـديدة للمصفوفة الأساسية.
السطران 7 و 8 يقومان بنسخ عــناصر المصـفوفة arr2 أي العناصـر القديمــة إلى المصفوفة الجـديد arr .
السطران 9 و 10 يتم فيهما التخلص من المصفوفة الاحتياطية بأكبر قـدر مـن الامان من خلال حذفها ثم إسنادها إلى الصفر ، وفي الحقيقــة نحـن تخلصـنا الآن من العـناصر القديمـة ولكن مع ملاحظة أننا أبقينـا القـيم فـي المصـفوفة الجديدة.
الآن نأتي إلى النسخـة الثانية من هذا التـابع ، وهـذه النسخــة تقـوم بـالحجز بشكل يدوي وليس بشكل آلي ، والاختلاف الوحيد بينها وبين النسخة الأولـى هـي فقط أن المتغير _capacity سـيكون محـدداً بـرقم معـين يحـدده الصـنف حسـب احتياجاتـه الخاصـة ولـيس بشـكل آلـي أي زيـادة المتغيـر _capacity بالعدد 30 .
هذا هـو تعريف هذا التابع:

```
1. template <class T>
2. void array<T>::allocater(int x)
3.          {
```

```
4.                  int m=_capacity;
5.                  arr2=arr;
6.                  _capacity=chapter(x);
7.                  arr=new T[_capacity];
8.                  for(int i=0;i<m;i++)
9.                        arr[i]=arr2[i];
10.                       delete[] arr2;
11.                       arr2=0;
12.
13.                 }
```

لاحظ أنه لا اختلاف بين النسخة السابقة والنسخة الحالية من التابع allocater إلا في السطر 4 و السطر 6.

عليك أن تتأكد أنني حينما أقول النسخة الأولى من التابع والنسخة الثانية من نفس التابع فلا أعني أن هناك نسخة قديمة أو نسخة جديدة بل أعني أنه تمت زيادة تحميل التابع.

المعامل ( ):

الآن سنأتي إلى تعريف المعاملات في هذا الصنف ، كما قلنا سابقاً حينما تود زيادة تحميل معامل ما ضمن صنف فإنه لا قواعد لفعل ذلك بل فقط كتابة الكلمة المفتاحية operator ، وأن عليك أن تحدد الغرض من المعامل وما هي الآثار التي ستطرأ على الصنف بعد أن يقوم بمهمته وماهي القيمة المعادة له.

سنقوم بزيادة تحميل المعامل ( ) ، حتى يصبح قادراً على الحلول مكان تابع البناء ، لن يكون بديلاً عن تابع البناء بل سيكون قادراً على فعل الأثر نفسه الذي يقوم به تابع البناء ، فسيكون قادراً على إعادة الصنف إلى وضعه الافتراضي ، وسيكون بإمكان مستخدم الصنف ، إعادة استخدام الصنف وفق ذاكرة محدد يعينها هو.

القيمة المعادة لهذا التابع ستكون من النوع void ، والسبب لذلك هو أن أثره سيكون داخل الصنف ولن يتفاعل مع كائنات أخرى من نفس النوع أو من أنواع أخرى.

انظر إلى تعريف هذا المعامل:

```
1. template <class T>
2. void array<T>::operator()(int m)
3.          {
4.
5.                  arr2=arr;
6.                  _capacity=chapter(m);
7.                  _size=m;
8.                  arr=new T[_capacity];
9.                  for(int i=0;i<_capacity;i++)
```

```
10.                        arr[i]=0;
11.                        delete []arr2;
12.                        arr2=0;
13.
14.                  }
```

انظر إلى رأس التابع في السطرين 1 و 2.
يتم نسخ المصفوفة الأساسية إلى الاحتياطية في السطر 5 ، والسبب في ذلك هو أننا سنقوم بإلغاء ذاكرة المصفوفة الأساسية عن طريق التعامل مع نفس العنوان الذي تشير إليه لكن سيتم جعل المصفوفة الاحتياطية هي التي ستقوم بالإلغاء ولا سبب ذلك فلو جعلنا المصفوفة الأساسية هي التي تقوم بالحذف لما وجد أي مشكلة ولكن تم اتخاذ هذا الإجراء لزيادة الإطمئنان أنه لن تحدث كوارث حينما يتعامل الصنف مع محتويات كبيرة نسبياً.
في السطر 6 يتم جلب حجم الذاكرة التي سنحجزها للحاوية عبر التابع chapter وإسنادها إلى المتغير _capacity .
في السطر 7 يتم إسناد العدد الممرر إلى هذا المعامل ، والذي هو الحجم الجديد للحاوية إلى المتغير _size .
في السطر 8 ، يتم حجز الذاكرة للحاوية بشكل جديد.
السطران 9 و 10 يقومان بتهيئة عناصر الحاوية الجديدة بالرقم 0 لأغراض الأمان ليس إلا.
السطران 11 و 12 يتم فيها التخلص من المصفوفة الاحتياطية أو الذاكرة القديمة بشكل آمن.

معامل الإسناد (=):

يعتبر هذا المعامل أحد الأدوات المهمة إذا ما أردت للصنف الذي تقوم بإنشاءه أن يتوسع أكثر ويتعامل مع كائنات من أنواع أخرى وليس من كائنات من نفس النوع ، علينا أولاً أن نحدد الوسائط التي سيأخذها هذا المعامل ونوع القيمة المعادة وما هو عمله على الصنف .
كما تعلم فإن هذا الصنف يأخذ وسيط واحد لا زيادة أو أقل وهو الصنف الذي تريد إسناده ، وهو في هذه الحالة عبارة عن حاوية من نفس النوع ، أما القيمة المعادة فهي بديهياً حاوية أو نفس الحاوية أو الكائن الذي ستتم عملية الإسناد إليه .
عليك دائماً التفكير قبل زيادة تحميل أي معامل عما تريد أن يفعل بالصنف ونوع الوسائط وما إلى ذلك.
انظر إلى تعريف معامل الإسناد:

```
1. template <class T>
2. array<T>& array<T>::operator=(array<T> &rhs)
3.          {
4.                _size=rhs.size();
5.                _capacity=rhs.capacity();
6.                int i=0;
7.                alloce ();
8.                for( i=0;i<_capacity;i++)
9.                      arr[i]=rhs[i];
```

```
10.                     return *this;
11.               }
```

انظر إلى رأس التابع في السطرين 1 و 2 .
في السطر 4 يتم إسناد حجم الصنف أو الحاوية التي قمنا بتمرير إلى حجم الصنف الأساسي.
نفس الأمر يحدث في السطر 5 بالنسبة للمتغير الاهم _capacity ، وبما أن المتغير _capacity في الصنف الممرر مقسم وفق تقسيمنا فلن نحتاج إلى تابع التقسيم chapter .
في السطر 7 يقوم الصنف باستدعاء التابع الداخلي alloce( ) ، حيث أن وظيفة هذا التابع هو القيام بعملية تخصيص وإعادة تخصيص جديدة للذاكرة ويعتمد في ذلك على المتغير _capacity والذي هو حالياً نفس المتغير _capacity في الصنف الممرر.سنصل إلى تعريف هذا التابع حينما ننتهي من شرح هذا المعامل.
في السطرين 8 و 9 يتم إسناد جميع عناصر الحاوية الممررة إلى المصفوفة الأساسية في الحاوية وهذه المرة ستستمر حلقة for حتى العنصر الأقل من _capacity وليس من _size ، والسبب واضح طبعاً.
السطر 10 يقوم بإعادة الكائن الذي استدعى المعامل = ، بواسطة إنشاء إشارة للمؤشر this .

التابع alloce( ) :

هذا التابع من العمليات الداخلية للحاوية ولن يكون أبداً من الواجهة ووظيفته هي القيام بعمليات تخصيص وإعادة تخصيص للذاكرة وتصفير جميع أعضاء الحاوية والسبب في ذلك حتى تكون الحاوية فارغة من العناصر وبالتالي تكون مستعدة لملأها من حاوية أخرى من نفس الصنف بواسطة نفس معامل الإسناد أو الإلحاق = .
انظر إلى تعريف هذا التابع:

```
1. template <class T>
2. void array<T>::alloce ()
3.         {
4.               delete [] arr;
5.               arr=0;
6.               arr=new T[_capacity];
7.               for (int i=0;i<_capacity;i++)
8.                     arr[i]=0;
9.         }
```

انظر إلى رأس التابع في السطرين 1 و 2.
في السطران 4 و 5 يتم التخلص من الذاكرة الأساسية وإلغاؤها وحذفها بطريقة أمنى.
يتم حجز ذاكرة جديدة للحاوية في السطر السادس بنفس حجم الذاكرة السابقة.
السطران 7 و 8 يقومان بعملية إسناد جميع عناصر الحاوية إلى الصفر.

المعامل [ ] :

يجب على هذا التابع أن يعمل على حل مشكلة الخروج خارج حدود الحاوية، هل تتذكر المصفوفة وما الذي سيحدث لها إذا خرجت خارج حدودها ، خاصة أنها لن تشتكي بأي شيء إلا بعد مرور وقت طويل حينما ينهار البرنامج الذي تقوم أنت بكتابته ، لذلك على هذا المعامل أن يحل هذه المشكلة جذرياً من خلال تخصيص ذاكرة جديدة للحاوية وليس بالإشارة إلى عنصر غير موجود في الحاوية.

هذا المعامل لا يعيد نفس الحاوية ولكنه يعيد عنصر من أحد عناصر هذه الحاوية أما عن الوسائط التي يستقبلها هذا المعامل فهو وسيط واحد من النوع int ، وهو فهرس العنصر الذي يريد المستخدم إيجاده.

على هذا المعامل أيضاً التعامل مع مشكلة أن يطلب المستعمل عنصر غير موجود إما لأن فهرسه أقل من الصفر أو لأن فهرسه أكبر من _capacity أو حتى _size .

باختصار على هذا المعامل في بعض الحالات الاستثنائية أن يقوم بعملية تخصيص وإعادة تخصيص جديدة للذاكرة.

انظر إلى تعريف هذا المعامل:

```
1. template <class T>
2. T& array<T>::operator[](int x)
3.          {
4.               if(x>_size){
5.                    if (x<_capacity){_size=x+1; return arr[x];}
6.                    else  {allocater(x);_size=x+1;return arr[x];}
7.               }
8.               else if(x<0) return arr[0];
9.               else return arr[x];
10.              }
```

انظر إلى رأس التابع في السطرين 1 و 2 ، تلاحظ أن الوسيط الممرر هو المتغير x من النوع int .

السطر 4 يسأل الصنف إن كان العدد الممرر أكبر من حجم الحاوية أي المتغير _size ، في حال كان كذلك فهنا ندخل في إحدى أخطر الحالات ألا وهي التأشير خارج حدود المصفوفة arr فلو سمحنا أن يعيد هذا التابع الفهرس دون أي تأكد فسيكون بالفعل هناك حالات لخروج خارج حدود المصفوفة الديناميكية arr .

في حال نجاح السطر 4 يدخل البرنامج في جملة if أخرى وهي هذه المرة سؤال الصنف إن كان العدد الممرر أصغر مما تستطيع الحاوية استيعابه وفي حال كان كذلك تعيد الحاوية العنصر من المصفوفة arr وتقوم برفع المتغير _size إلى نفس العدد الممرر زائداً واحداً وهو نفس الذي يحدث في المصفوفات الحقيقية.

أما في حال كان رقم الفهرس غير موجود أصلاً في الحاوية أو أكبر مما تستطيع الحاوية استيعابه فسينتقل التنفيذ إلى السطر 6 حيث يقوم الصنف باستدعاء التابع العضو allocater وتمرير رقم الفهرس إليه وبالتالي عملية تخصيص وإعادة تخصيص جديدة ، حيث يتم تقسيم جديد للذاكرة بواسطة

التابع chapter وحالما ينتهي البرنامج من هذه العملية فسيتم رفع المتغير _size إلى نفس قيمة الفهرس (المتغير x) مضافاً إليها واحد.
أما في حال أنه أصلاً لم ينجح اختبار الجملة if في السطر 4 وبالتالي ليس لدينا فهرس أكبر من حجم الحاوية فسينتقل التنفيذ إلى السطر 8 وهو يتعامل مع مشكلة أخرى من مشكلات التأشير خارج حدود المصفوفة ولكنها هذه المشكلة اكبر حيث يتعامل مع الإدخالات القاتلة مثل طلب الفهرس -1 ، هذا الفهرس غير موجود وهذه المشكلة ليس لها حل أصلاً لذلك يقوم الصنف بإعادة أول عنصر في الحاوية. بإمكانك التعامل مع هذا الخطأ على أنه استثناء وربما قد تريد تطوير الحاوية لتصبح قادرة على التعامل مع الاستثناءات .
أما في حال أنه لم يكن هناك أصلاً أي عملية غير شرعية خارج حدود الحاوية فسينتقل التنفيذ إلى السطر 9 حيث يقوم المعامل بإعادة العنصر المراد دون أية مشاكل.

التابع ( )find :

هذا التابع هو أحد الخدمات المتطورة التي تقدمها الحاوية حيث يبحث ضمن عناصره عن قيمة محددة أو معينة ويعيد رقم الفهرس الذي يكون العنصر موجود من ضمنه.
هذا التابع يستقبل عنصر من نفس العناصر التي تحتويها الحاوية ويقوم بإعادة رقم الفهرس والذي هو من النوع int .
هذا هو تعريف هذا التابع:

```
1. template <class T>
2. int array<T>::find (T x)const
3.          {int i=0;
4.               for( i=0;i< _size;i++)
5.                    if (x==arr[i]) return i;
6.          return -1;
7.          }
```

أنظر إلى رأس التابع في السطرين 1 و 2 .
يتم البحث بواسطة المعامل [ ] في السطرين 4 و 5 ، وكما ترى فهذا البحث هو نفسه طريقة البحث التي تناولنها في وحدة المصفوفات.
وبالطبع في حال إذا لم يجد التابع أي شيء أو العنصر المراد فإنه يقوم بإعادة الرقم -1 للدلالة على أنه لم يستطع إيجاد العنصر المراد.

التابع ( )clean :

يقوم هذا التابع بوظيفة مهمة للغاية وهي تنظيف الحاوية ومسحها مسحاً تاماً ، وإعادتها إلى وضعها الافتراضي دون وجود أي عناصر أو أي حجم.
أنظر إلى تعريف هذا التابع:

```
1. template <class T>
2. void array<T>::clean()
3.          {
4.               arr2=arr;
```

```
5.                  _capacity=30;
6.                  _size=0;
7.                  arr=new T[_capacity];
8.                  for(int i=0;i<_capacity;i++)
9.                  arr[i]=0;
10.                       delete []arr2;
11.                       arr2=0;
12.                 }
```

طريقة مسح التابع clean لجميع عـناصر الحاوية هـي طريقة شبيهه بـالطرق التي تقوم بها بعض التوابع الأعضاء وبالتالي فهي لا تحتاج لأية شرح.

التابع ( )erase :

وظيفـة هذا التابع خطيرة نـوعاً ما فهـو يقوم بحذف عــنصر ربمـا مـن منتصـف الحاوية وليس من آخرهـا ، الخوارزميـة التـي يعــمل بهـا هـذا التـابع بسـيطة نـوعاً ما ، حيث يقوم بأخذ العــنصر الـذي يكـون بعـد العـنـصر المـراد مسحــه ويسنده إلى العـنصر المراد مسحـه ثم يقوم بأخذ العـنصر الذي يكـون التـالي بعـد العـنصر المسند ويسـنده إلـى العنصـر الـذي يكـون بعـد العــنصر المـراد مسحـه ، العـملية غير مفهـومـة ولكن أنظر إلى هذه المصفوفة.

74	93	50	10	20	12

نريد حذف العـنصر رقم 2 في المصفوفة والذي هـو في هذه الحالـة العــنصر 10 ، كـون فهرس المصفوفة يبدأ من الصفر وليس الواحد.
أنظر إلى مالذي سيحدث لهذه المصفوفة.

تم إلغاؤه	74	93	50	20	12

انتقلت العـناصر التي بعـد العـنصر المراد حذفـه إلـى مرتبـة أقـل أمـا المكـان الأخير من الذاكرة وهـو مكان رقم 5 فتم حذفـه من الذاكرة.
هذه هـي العـملية التي سيقوم بها التابع ( )erase .
أنظر إلى تعريف هذا التابع:

```
1. template <class T>
2. void array<T>::erase(int x)
3.          {
4.                  if ((x>_size)||(x<0)) return;
5.                  _size=_size-1;
6.                  arr[x]=0;
7.                  for(int i=x;i<_size;i++)
8.                        arr[i]=arr[i+1];
9.
10.                 }
```

الوسيط الممرر لهذا التابع من النوع int وهـو رقـم العنصـر المـراد حذفـه كمـا يظهر في رأس التابع في السـطرين 1 و 2 .
السطر رقم 4 يتأكد إن كان المستخدم يريد حذف عـنصر غير موجود أصلاً في الحاوية إما لأنـه رقم فهرسـه أصغر مـن الصـفر أو لأن رقـم فهرسـه أكبـر مـن حجم الحاوية وفي حال حدوث أي من هذين السببين فإنه يرجع دون أن يعيـد أي قيمة.
السطر 5 يقوم بإنقاص حجم الحاوية عـدداً واحداً فقط.
تتم عـملية انتقال العـناصر التي بعـد العـنصر المحـذوف إلـى فهارسـها التـي أصبحت أقل بعدد واحد عـن المرات السابقة وذلك في السـطرين 7 و 8 .
وهـكذا ينتهي التابع erase .

معامل الجمع + :

تعريف هذا المعامل يعتبـر صـعباً بعـض الشـيء وطريقـة عـمله تعتبـر أيضـاً حيوية نوعاً ما لهذا الصنف ، يقوم هذا الصنف بدمج الحاويتين المـراد جمعهمـا وإعادة حاوية أكبر تضم هـاتين الحـاويتين السـابقتين ، مـن الملاحـظ هـنـا أن هذه العـملية لن تكون إبدالية بشأن دمج حاويتين لأنـه لـن يمكنـك فعـل ذلـك حتى لو أردت ، فعملية دمج حاويتين سينتج عـنها حاوية أكبر حجماً العــناصر الأول ستضم فيها الحاوية الأولى والعــناصر الاخيـرة ستضـم الحاويـة الثانيـة أي أنها لن تكون إبدالية.
يستقبل هذا التابع كبارامتر له حاوية كاملة ، ويقوم بإعادة حاوية أخرى.
أنظر إلى تعريف هذا التابع:

```
1. template <class T>
2. array<T> array<T>::operator+ ( array<T>& rhs)
3.         {
4.                 int i=_size+rhs.size();
5.                 array<T> a(i);
6.                 for(int j=0;j<_size;j++)
7.                         a[j]=arr[j];
8.                  int k=0;
9.                 for(k=0,j=j;j<i;j++,k++)
10.                                a[j]=rhs[k];
11.                        return a;
12.                }
```

أنظر إلى رأس التابع في السـطرين 1 و 2.
في السطر 4 تم الإعلان عـن المتغير i والذي سنقوم بجمع حجم الحاوية الاولى والحاوية الثانية وإسناد القيمة إليه ، وبالتالي فإن المتغير i سيكون حجم الحاوية الجـديدة الناتجـة عـن عملية الجمع.
في السطر 5 تم الإعلان عـن الحاوية a والتي سيتم حجز ذاكرة لها بمقدار المتغير i .
في السطرين 6 و 7 يتم أخذ جميع قيم عـناصر الحاوية الأولى (الحاوية التي استدعت معامل الجمع) ووضعها في العـناصر الأولى من الحاوية a .
في السطرين 9 و 10 يتم أخذ جميع عـناصر الحاوية الثانيـة (الحاوية التي هي حالياً بارامتر أو وسيط) ووضعها في العـناصر الأخيرة من الحاوية a .

لاحظ كيف تتم إسناد عناصر الحاوية الثانية إلى الحاوية الأولى ؛ تجد أن الحلقة for لم تبدأ الإسناد إلى الحاوية a من الصفر بل من الفهرس الذي توقف فيه الحلقة for الأولى أو السابقة.
في السطر 11 يتم إعادة الحاوية a .

التابع save( ) :

تركنا لك فرصة تطوير هذا التابع حتى يصل إلى درجة مرضية أما عن التابع الموجود في هذا المثال فهو بدائي نوعاً ما ويحتاج للتعامل مع بعض الحالات.
ربما أيضاً قد تود اعتبار أن مهام حفظ الملفات وتحميلها ليس من مهام الحاوية بل من مهام العناصر الموجود في الحاوية ، ليس في الأمر قاعدة أو طريقة معينة بل الأمر يرجع في أغلب الحالات إلى المبرمج ووجهة نظره فحسب.
أنظر إلى تعريف هذا التابع:

```
1. template <class T>
2. void array<T>::save()
3.         {
4.                 ofstream a("file",ios::binary);
5.                 for(int i=0;i<_size;i++){
6.                 a.write( (char*) &arr[i], sizeof T );
7.                 }
8.                 a.close();
9.         }
```

أنظر إلى رأس التابع في السطرين 1 و 2 ، ربما في المرة القادمة قد تود جعله يستقبل اسم الملف كبارامتر له.
في السطر 4 يتم إنشاء كائن ofstream لإخراج البيانات أو حفظ عناصر الحاوية فيه ويتم إنشاء ملف اسمه file ويتم فتحه على هيئة ثنائية في الأسطر من 5 إلى 7 يتم حفظ جميع عناصر الحاوية في المصفوفة بواسطة الحلقة for .
في السطر 8 يتم إغلاق هذا الملف.

التابع load( ) :

يقوم هذا التابع بأخذ البيانات من الملفات أو العناصر ووضعها في الحاوية ، لا تقلق من كيفية حجز الذاكرة فالحاوية التي قمنا بكتابتها قادرة على التعامل مع هذه الحالات المقلقة فهي مستقرة لدرجة تمنع الخطر عنها عند التعامل مع الملفات وحجز الذاكرة المناسبة للعناصر الجديدة الآتية من ملف ما.
أنظر إلى تعريف هذا التابع:

```
1. template <class T>
2. void array<T>::load(){
3.                 ifstream a("file",ios::binary);
4.                 int j=sizeof T;
5.                 int i=0;
```

```
6.                  while (!a.eof()){
7.                  a.read( (char*) &arr[i], sizeof T );i++;
8.                  }
9.                  a.close();
10.                       _size=i-1;
11.                 }
```

انظر إلى رأس التابع في السطرين 1 و 2 .
في السطر الثالث يتم إنشاء أحد كائنات القراءة ifstream والذي سيقوم بفتح الملف file على هيئته الثنائية وليس النصية.
في السطرين 6 و 7 و 8 يتم التعامل مع الملف من خلال قراءة جميع العناصر وإسنادها إلى العناصر الفارغة في الحاوية.
ربما في المستقبل قد تود أن تقوم بتطوير هذا التابع حتى يستطيع مستخدم الصنف وضع اسم الملف الذي يود تحميل البيانات منه ، أيضاً وكما ترى فهناك بعض الثغرات الخطيرة في هذا التابع وقد يجعل من الصنف ينهار في بعض الحالات ، فهذا التابع حينما قمت بكتابته افترضت أن المستخدم يريد فحسب وضع البيانات الموجودة في الملف في الحاوية فحسب ولم أفترض إنه قد يقوم بوضع البيانات في حاوية قد تكون ممتلئة وليست فارغة كما افترضت ، قد ينشأ عن حالات الاستعمال هذه أخطاء خطيرة وصعبة الاكتشاف نوعاً ما.
تركت لك المجال حتى تقوم بتطوير الصنف ليصبح قادراً على التعامل مع جميع الحالات التي ذكرتها وقد توسع من مهامه ليصبح قادراً على الفرز وما إلى ذلك من أمور.

التوابع size( ) و capacity( ) :
ليس هناك من شيء لشرحه بالنسبة لهذه التوابع فهي توابع وصول فقط قد يستفيد منها المستخدم أو الصف نفسه في عملياته الداخلية كما رأينا سابقاً .
هذا هو تعريف هاذين التابعين.

```
1. template <class T>
2. int array<T>::size(){return _size;}
3. /*          capacity( )             */
4. template <class T>
5. int array<T>::capacity(){return _capacity;}
```

وعموماً ستجد هذا المثال موجوداً في المرفقات مع هذا الكتاب.

دالة الاختبار main( ) :
وضعت هذه الفقرة لتعريف بكيفية استخدام هذه الحاوية.

```
1. int main()
2. {
3.    array <int> a(4);
4.    for(int i=0;i<a.size();i++)
5.          a[i]=i*2;
```

```
6.
7.    array <int> b(5);
8.    for( i=0;i<b.size();i++)
9.          b[i]=i*4;
10.         array <int> c(40);
11.
12.
13.         c=b+a;
14.
15.         for( i=0;i<c.size();i++)
16.               cout << i << "::::\t\t" <<c[i] << endl;
17.
18.         cout << "(c):\n";
19.         cout << "size():::::::" << c.size() << endl;
20.         cout << "capacity():::" << c.capacity() << endl;
21.
22.         return 0;
23.         }
24.
```

استخدام الحاوية مع أصناف المستخدم:

بقي الآن التعليمات التي سنذكرها للمستخدم حتى يستطيع استعمال هذه الحاوية وما هي مواصفات الصنف حتى تستطيع الحاوية استعماله.

لا تعليمات كثيرة هنا بل فقط على المستخدم أن تكون المعاملات << و >> معرفة ضمن الصنف وإلا فإن الحاوية لن تعمل.

أيضاً على صنف المستخدم أن يكون المعامل = معرفاً ضمنه.

أيضاً لا بد من وجود تابع بناء النسخة حتى تعمل الحاوية بشكل جيد.

نفس المقاييس والمواصفات للكائنات إذا ما أردت استخدامها في مكتبة القوالب القياسية يجب أن تكون هي نفسها هنا.

الملحق (أ)

أسبقية المعاملات الحسابية

<table>
<tr><th>المستوى</th><th>المعاملات</th></tr>
<tr><td>1</td><td>::</td></tr>
<tr><td>2</td><td>( )   [ ]   ->   .</td></tr>
<tr><td>3</td><td>Sizeof & * -- ++ + - ~ !</td></tr>
<tr><td>4</td><td>* / %</td></tr>
<tr><td>5</td><td>+ -</td></tr>
<tr><td>6</td><td>>= > < <=</td></tr>
<tr><td>7</td><td>8= ==</td></tr>
<tr><td>8</td><td>&</td></tr>
<tr><td>9</td><td>|</td></tr>
<tr><td>10</td><td>&&</td></tr>
<tr><td>11</td><td>||</td></tr>
<tr><td>12</td><td>المعامل الشرطي ? :</td></tr>
<tr><td>13</td><td>= += -= *= /= %= |= <<= >>=</td></tr>
<tr><td>14</td><td>throw</td></tr>
<tr><td>15</td><td>,</td></tr>
</table>

الملحق (ب)

الكلمات المحجوزة في السي بلس بلس

الكلمات المحجوزة بالترتيب الألفبائي:

auto
break
case
catch
char
class
const
continue
default
delete
do
double
else
enum
extern
float
for
friend
goto
if
int
long
mutable
new
operator
private
protected
public
register
return
short
signed
sizeof

static
struct
switch
template
this
throw
typdef
union
unsigned
virtual
void
volatile
while

الملحق (ب)

المعالج التمهيدي

The Preprocessor

بداية:

حينما يبدأ المترجم عـمله فإن أول ما يعـمله هـو تشـغيل المعـالج التمهيـدي، والـذي يبحـث عــن الأوامـر الخاصــة بـه ، وكـل أمـر يقـوم المعـالج التمهيـدي بمعالجتـه سينتج عـنه تغيير في نص الاوامر المصدر.
الأوامر التي يبحث عـنها المعالج التمهيـدي تبـدأ برمـز الجنيـة # ، مثـل الأمـر include .

الأمر define والثوابت:

ربمـا اسـتخدمنا فـي أمثلـة هـذا الكتـاب الأمـر define ، هـذا الأمـر يسـتبدل سلسـلة الأحرف بالقيمـة الموضوعـة حسـب الأمـر وهـو لا يفحـص الأنـواع .
انظر إلى هذا السطر:

```
#define MAX 50
```

يحوي هذا الأمر توجيه للمترجم حيث يخبره أنك إذا وجدت أي سلسلة أحـرف MAX فقم باستبدالها بالرقم 50 ، فلو كتبت هذا الأمر مثلاً:

```
int arr[MAX];
```

فإنها ستظهر في الأوامر المصدر النهائيـة هـكذا:

```
int arr[50]
```

وليس بنفس الصيغـة التي كتبت بها.

توابع المعالج التمهيدي:

بإمكانك استخدام المعالج التمهيدي بدلاً عــن التوابـع فهــو أسـرع ولا يلزمـك بفحص الأنـواع ولا بالتحميل الزائد ولا بالقوالب ولا بأي شـيء آخر.
تذكر هذا النوع من التوابع لا يعيد أي قيمـة وإنما يستبدل الأماكن التي ذكـرت فيها اسم التابع بالقيمـة المطلوب استبدالها.
تذكر أيضاً أوامر المعالج التمهيدي يجب أن تكون في نفس السطر ، إذا كانـت في سطرين فسيستغني المترجم عـن السطر الثاني ويعتبره خطأ.
انظر إلى هذا المثال:

CODE

```
1. #include <iostream>
2. #define POWER(x) x*x
3. #define POWER3(x) x*x*x
4. using namespace std;
5.
6.
```

```
7. int main()
8. {
9.    int a=0,b=0;
10.
11.         cout << "Enter a:\t"; cin >> a;
12.         cout << "Enter b:\t";cin >> b;
13.
14.         cout << endl << endl;
15.
16.         cout << "power:\t\t" << POWER(a) << endl;
17.         cout << "power3:\t\t" << POWER3(b) << endl;
18.
19.         return 0;
20.   }
```

في السطر الثاني والثالث وبواسطة الأمر define تم تعريف تابعان اثنان الأول يقوم بتربيع العدد الممرر والتابع الثاني يقوم بتكعيب العدد الممرر.
يتم استخدام هذان التابعان في السطرين 16 و 17 ، لاحظ أن البرنامج لا يطبع القيمة المعادة بل يطبع القيمة المستبدلة ، فهو لا يعتبر POWER تابعاً بل رمزاً ، يجب استبداله بإحدى القيم.
هناك استخدامات كثيرة متقدمة للمعالج التمهيدي وخاصة في حالات مستويات اكتشاف الأخطاء ، ولكن الكتاب ركز على المبادئ الأساسية لأن الهدف من الكتاب هو إعطاؤك مقدمة كبيرة وواسعة للسي بلس بلس.

تمت بحمـــد الله

سلطان محمد خميس الثبيتي
sultan_altaif@yahoo.com
طالب في جامعة الطائف